2001-06

Analysis of tracking and identification characteristics of diverse systems and data sources for sensor fusion

Wilson, Dean A.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/1189

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ANALYSIS OF TRACKING AND IDENTIFICATION CHARACTERISTICS OF DIVERSE SYSTEMS AND DATA SOURCES FOR SENSOR FUSION

by

Dean A. Wilson

June 2001

Thesis Advisor:	Russ Duren
Thesis Co-Advisor:	Gary Hutchins

ANALYSIS OF TRACKING AND IDENTIFICATION CHARACTERISTICS OF DIVERSE SYSTEMS AND DATA SOURCES FOR SENSOR FUSION

Dean A. Wilson-Lieutenant Commander, United States Navy
B.S.A.E., Virginia Polytechnic and State University, 1990
Master of Science in Aeronautical Engineering –June 2001
Advisor: Russell Duren, Department of Aeronautical Engineering
Co-Advisor: Gary Hutchins, Department of Electrical Engineering

In the Command and Control mission, new technologies such as 'sensor fusion' are designed to help reduce operator workload and increase situational awareness. This thesis explored the tracking characteristics of diverse sensors and sources of data and their contributions to a fused tactical picture. The fundamental building blocks of any sensor fusion algorithm are the tracking algorithms associated with each of the sensors on the sensor platform. In support of this study, the MATLAB program '*fusim*' was written to provide acquisition managers a tool for evaluating tracking and sensor fusion algorithms.

The *fusim* program gives the user flexibility in selecting: sensor platforms, up to four sensors associated with that platform, the target types, the problem orientation, and the tracking algorithms to be used with the sensors. The *fusim* program was used to compare tracking algorithms in a multiple sensor/multiple target environment. Specifically, the Probabilistic Data Association Filter, the Interacting Multiple Models Filter, the Kalman Filter and the Constant Gain Kalman Filter were evaluated against multiple maneuvering, non-maneuvering, and fixed targets. It is recommended that this study be continued to evaluate advanced tracking and data association techniques, to expand the program to allow attribute tracking and identification, and to study the Human-Machine Interface aspects of sensor fusion.

DoD KEY TECHNOLOGY AREA: Sensor Fusion, Modeling and Simulation, Sensors, Command Control and Communications, Computing and Software

KEYWORDS: Data Fusion, Sensor Fusion, Tracking, Tracking Algorithms, Kalman Filter, Probabilistic Data Association, PDA, Interacting Multiple Models, IMM, Simulation

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.	

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters) Analysis of Tracking and Identification Characteristics of Diverse Systems and Data Sources for Sensor Fusion	**5. FUNDING NUMBERS**
6. AUTHOR Dean A. Wilson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) NAWC-AD PMA-231 47123 Buse Rd Unit IPT Bldg 2272 Suite 455 Patuxent River, MD 20670	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.	
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

In the Command and Control mission, new technologies such as 'sensor fusion' are designed to help reduce operator workload and increase situational awareness. This thesis explored the tracking characteristics of diverse sensors and sources of data and their contributions to a fused tactical picture. The fundamental building blocks of any sensor fusion algorithm are the tracking algorithms associated with each of the sensors on the sensor platform. In support of this study, the MATLAB program '*fusim*' was written to provide acquisition managers a tool for evaluating tracking and sensor fusion algorithms.

The *fusim* program gives the user flexibility in selecting: sensor platforms, up to four sensors associated with that platform, the target types, the problem orientation, and the tracking algorithms to be used with the sensors. The *fusim* program was used to compare tracking algorithms in a multiple sensor/multiple target environment. Specifically, the Probabilistic Data Association Filter, the Interacting Multiple Models Filter, the Kalman Filter and the Constant Gain Kalman Filter were evaluated against multiple maneuvering, non-maneuvering, and fixed targets. It is recommended that this study be continued to evaluate advanced tracking and data association techniques, to expand the program to allow attribute tracking and identification, and to study the Human-Machine Interface aspects of sensor fusion.

14. SUBJECT TERMS Data Fusion, Sensor Fusion, Tracking, Tracking Algorithms, Kalman Filter, Probabilistic Data Association, PDA, Interacting Multiple Models, IMM, Simulation	**15. NUMBER OF PAGES** 201
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ANALYSIS OF TRACKING AND IDENTIFICATION CHARACTERISTICS OF DIVERSE SYSTEMS AND DATA SOURCES FOR SENSOR FUSION

Dean A. Wilson
Lieutenant Commander, United States Navy
B.S.A.E., Virginia Polytechnic and State University

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN
AERONAUTICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL
June 2001**

Author: [illegible]
Dean A. Wilson

Approved by: [illegible]
Russ Duren, Thesis Advisor

[illegible]
Gary Hutchins, Co-Advisor

[illegible]
Max Platzer, Chairman
Department of Aeronautics and Astronautics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the Command and Control mission, new technologies such as ‘sensor fusion’ are designed to help reduce operator workload and increase situational awareness. This thesis explored the tracking characteristics of diverse sensors and sources of data and their contributions to a fused tactical picture. The fundamental building blocks of any sensor fusion algorithm are the tracking algorithms associated with each of the sensors on the sensor platform. In support of this study, the MATLAB program ‘*fusim*’ was written to provide acquisition managers a tool for evaluating tracking and sensor fusion algorithms.

The *fusim* program gives the user flexibility in selecting: sensor platforms, up to four sensors associated with that platform, the target types, the problem orientation, and the tracking algorithms to be used with the sensors. The *fusim* program was used to compare tracking algorithms in a multiple sensor/multiple target environment. Specifically, the Probabilistic Data Association Filter, the Interacting Multiple Models Filter, the Kalman Filter and the Constant Gain Kalman Filter were evaluated against multiple maneuvering, non-maneuvering, and fixed targets. It is recommended that this study be continued to evaluate advanced tracking and data association techniques, to expand the program to allow attribute tracking and identification, and to study the Human-Machine Interface aspects of sensor fusion.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge the financial support of PMA-231 and the Office of Naval Research for allowing this research to be conducted at the Naval Postgraduate School under Contract N0042101WR02886. A special thank-you is included for Captain Lee Lilly, Commander Jim Clifton, Dr. Rabinder Madan, and Mr. Bill Hamel for helping to facilitate this effort.

I am greatly appreciative of the continued support of my co-advisors on this project, Professor Russ Duren and Professor Gary Hutchins. Thank-you for allowing me the flexibility I needed in my studies and schedule, and for the freedom to create this simulation in my own image.

I'd like to thank my kids, Jennifer, Stephanie, and Brian for their willingness to try new things, and their eagerness to achieve. You have kept me inspired!

I am truly grateful to my wife, Debbie, for her unwavering encouragement and support, (and for packing up the kids and the dog and following me all the way to California). Thank-you for lending me your ear at all hours of the day and night, and for never missing the chance to remind me that *I asked* to come here. I love you.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The mission of airborne command and control is much more than the advertised "airborne early warning (AEW)." The new buzzword for the E-2C Hawkeye Command and Control (C^2) mission is *Battle Management.* New technologies are being designed to help reduce operator workload and increase situational awareness. Ideally, these new capabilities will help tactical aircrews manage Battle Group assets and ultimately, the entire battlespace. The E-2C aircrew performs functions that cannot be replicated by a machine, that is, making battlespace assessments and taking charge of situations. The aircrew must have a thorough understanding of the tactical picture to formulate appropriate responses. The industrious aircrew will endeavor to formulate the "Big Picture." They mentally fuse all available information to create sufficient situational awareness (SA), to understand the tactical picture, and to decide on courses of action. However, situational awareness is easily disrupted by immediate threats, aircraft emergencies, or information overload. It is left up to the diligent aircrew to achieve and maintain SA.

Initial detection of targets within the E-2C aircraft occurs with the radar, the IFF detection system, and the Passive Detection System (PDS). These systems are coupled with an automatic tracking system within the mission computer. Secondary sources of information consist of the onboard data links including, CEC, JTIDS, Link-11, and Link-4. Other data sources include SATCOM data and information received via voice communications (UHF SATCOM, HF/VHF/UHF LOS). Additionally, the aircrew receives data such as the Air Tasking Order, Special Instructions, Airspace Restrictions, Rules of Engagement, Battlefield Situation Reports, and intelligence briefings prior to takeoff. All available information is used by E-2C aircrew to formulate battlespace assessments, and to create and maintain situational awareness. In a broader sense, the objective of any military engagement is to defeat the enemy with decisive, overwhelming force. For our military forces to take full advantage of the information and observations available, they must be able to interpret their battlespace and make timely and relevant decisions.

Warfare technology has continued to advance, and the battlespace has become much more complicated. Furthermore, the E-2C aircrew has been increasingly tasked with a wider variety of more demanding missions. To keep pace with this evolutionary cycle of weapons advances and more complicated missions, the E-2C weapon system is continuing to improve. Current weapons system improvements for the E-2C include an upgraded Electronic Surveillance Measures (ESM) system, a new mission computer and displays, the Cooperative Engagement Capability (CEC) system, and a Satellite Communications (SATCOM) data system. Future upgrades include the Radar Modernization Program (RMP), the UHF Electronically Scanned Array (UESA) antenna, the Surveillance Infrared Search and Track (SIRST) system, and the E-2C Multiple Source Integration system. These weapons system improvements are designed to improve the detection and processing of tactical information to more fully support the mission and ultimately the Battle Group Commander.

B. PURPOSE

This thesis explored the tracking characteristics of different sensors and sources of data and their contributions to a “fused” tactical picture. This ‘fusion’ aspect implies that one track will represent each contact of interest in the battlespace with all available position, identification and intentions data displayed for each track. How can ‘sensor fusion’ help with formulating a tactical picture? What are the unique attributes of the different sensors and data sources that can be combined to produce an accurate tracking picture? Finally, how can the different data sources be combined to best exploit the attributes of identification and position information? The purpose of this thesis was to explore the military application of Sensor Fusion technology by creating a MATLAB-based multiple-sensor, multiple-track simulation, and exploring some of the applicable tracking and fusion algorithms relevant to current and future sensor fusion capabilities.

C. TOPICS RELEVANT TO SENSOR FUSION

1. Definitions

First, a few definitions are necessary to ensure that no terminology problems are encountered throughout the remainder of this paper. Currently, sensor fusion takes many forms, whether from industry, government services, or military applications. The primary focus of this list is the military application of sensor fusion.

<u>Battlespace</u>

The term “battlespace” includes all of the area (land, sea, air, and space) of interest, all of the sensors and information sources that can contribute to a coherent tactical picture, all of the forces (friend, enemy, and neutral), weapon systems, and all communications systems in the area of interest.

Kinematic Data

Target kinematics are usually represented by the position, course, speed, altitude, or accelerations and turn/climb rates [Ref. 1].

Attribute Data

The target attributes are the features of the target such as returned signal strength, IR or radar signature, or target classification and ID information [Ref. 1].

State

The target state [Ref. 2] is a determination of target position, course, and speed, depending on the current sensor measurement and the previous state estimate. The state vector usually consists of the target position coordinates and the corresponding velocity components. The target state can also hold estimates of turn rates or other kinematic data.

Local Active Track

A local active track (LAT) refers to a track held by a sensor that is organic to the sensor platform. A track [Ref. 1] is the result of a state prediction from the tracking algorithm, and is based on the measurements from the sensor.

Similar Source Integration

Similar source integration (SSI) is the association and correlation of tracks from the same kind of sensors such as radar or IFF detection systems [Ref. 4].

Dissimilar Source Integration

Dissimilar source integration (DSI) is the association and correlation of tracks from multiple types of sensors such as an IFF-to-radar correlation or a LAT-LAT/Data Link-LAT correlation. A DSI must include the sensor registrations for time corrections and coordinate transformations [Ref. 4].

Multiple Source Integration

The multiple source integration (MSI) process performs the data association and correlation of tracks from multiple types of sensors. The output of the MSI process is a database of all possible track-to-track associations [Ref. 4].

Data Fusion

The most recent definition of "Data Fusion" is from the Joint Directors of Laboratories (JDL) paper of 1998. According to the JDL [Ref. 3], "Data Fusion" is the process of combining data to refine state estimates and predictions.

Sensor Fusion

Strictly speaking, the term sensor fusion is the combination of estimates from multiple sensors for the formulation of multiple tracks [Ref. 1]. The term "Sensor Fusion" is thrown around loosely and is sometimes taken to mean the same as the 'data fusion' term. While data fusion is truly a more encompassing term, the 'sensor fusion' term implies a distinct relationship between the measurements/observations of the battlespace sensors and the fusion of the resulting information. Therefore, 'sensor fusion' will be used throughout the remainder of this paper.

2. Combat Identification

Identification is the key element in the decision making process, i.e., the identification of a contact leads to a decision. Given the identification of a contact, the decision-maker can decide whether to investigate, monitor, ignore, intercept, or shoot the contact. For example, if a contact is flying in a zone where no friendly aircraft are known to be flying (with 100% certainty), the specific identity or intentions of the target may be irrelevant and still satisfy the conditions of combat identification (CID). Deciding which contacts to process (and when) is a function of the threat (situation) assessment and the assets available to meet the threat (impact assessment). Situation and impact assessment will be discussed in the next chapter.

The military application of sensor fusion is primarily the fusion of all organic sensor track data, off-board data link information, and archived information to form a single fused track for each contact of interest in the area of interest. In the CID process, three vital pieces of information about a contact are its position, its identification, and its intentions. Depending on the source of information, these attributes can be closely related, such as a radar track with IFF information fused by a tracking system, or as diverse as an ESM line of bearing that cuts through a postulated ground emitter site. The diversity of this data is precisely the purpose for studying the techniques of sensor fusion. The use of ID information leads to the ultimate goal of combat identification (CID) for weapons system engagement of enemy targets.

3. The Operator's Point of View

The goal of sensor fusion is to ease the workload and improve the situational awareness of the crew. Sensor Fusion systems are meant to aid the crew in the decision making process. In other words, let the machines do the number crunching, and let the humans make the decisions. One of the most difficult questions to answer is: What does the operator want? Unfortunately, an old adage still applies: If 20 different operators are asked for their views on how information should be displayed, the result will be 20 different answers. The operator is the primary interface between the display and the command and control decision tree. In the case of the E-2C, each operator can act independently.

With respect to sensor fusion, the individual display of information must address the diversity of missions within the E-2C or any other command and control platform. How the information is displayed is vitally important. The display of information can be prioritized depending on the tactical situation and the tactical impact of the target. The operator that is controlling a surface search coordination (SSC) mission does not have the same information and threat display priorities as the operator who is providing tactical control for an overland strike mission, even though both events may be happening simultaneously. Instead, the operators need maximum flexibility for prioritizing their individual display requirements for information, threats, timelines, etc. For the airborne battlefield command and control mission, displaying all mission-applicable information would be senseless for the user and impossible to fully interpret. Instead, the display of information should reflect a relative importance to the mission at hand, the desires of the operator, and the needs of the battlespace commander. Ideally, only those threats that are most applicable to the operator’s mission would be displayed.

D. CHAPTER OUTLINE

Chapter II is a discussion of Sensor Fusion and why it is important to the military. Chapter III is a description of modern tracking algorithms. Chapter IV is a detailed description of the MATLAB based fusion simulator program, including the evaluation parameters of the test sets. Chapter V describes the results of testing the tracking algorithms for stability, and presents the results of running several different test cases. Conclusions and recommendations are included in Chapter VI.

II. OVERVIEW OF SENSOR FUSION

A. WHY SENSOR FUSION?

Simply put, tactical aircrews have been doing sensor fusion all along by achieving situational awareness with respect to the tactical picture. This thesis was written with the platform and the operator as primary concerns. The importance of the operator and the platform cannot be overstated: every platform has unique sensor capabilities and attributes, communications systems, and display capabilities. And every operator has unique display requirements, depending on the platform, the sensors and data sources, and the mission being conducted. The sensor/comm suite and display capabilities of the AEGIS cruiser and the E-2C are vastly different, yet the C^2 goal is still the same. The operator must be able to interpret and respond to the tactical situation depending on his/her mission goals. Accurate tracking and identification are paramount to this process. Primarily, the operator would like to know the identification of a contact so the tactical impact can be determined. The operator must consider the impact to his platform, the platforms under his control, the carrier, and ultimately, the entire battle group. This is how Combat Identification becomes an important by-product of the sensor fusion ID process. If engaging a contact is within the ROE, the operator must be able to allocate assets quickly and decisively without distractions from information overload, false tracks, or dual tracks.

Instead of cluttering up the scope with thousands of reports, the information would ideally be sorted by target type, registered in time to allow a common reference time, and fused with other reports and local active tracks. This data association provides an identification of a track, or increases the confidence of the position report. Even if there is no association with a local track, such as in the case of a known SAM site, the information must be displayed in a timely and relevant manner. Additionally, the information could be used to cue the operator to look for contacts in an area where no active track exists, or to cue the radars of other platforms to search the area. Contact reports that are considered an immediate threat should pop up and alert the operator so that the operator can respond appropriately. The display of threat information should coincide with the operator's display desires and the tactical situation. For example, certain high-priority threats can be displayed anytime, anywhere on the scope. Other threats/tracks may only be displayed in an area of interest, depending on the mission or the needs of the operator.

B. THE LEVELS OF SENSOR FUSION

The Joint Directors of Laboratories (JDL) data fusion process model is the most commonly used model to communicate ideas about algorithms, systems, and research (Figure 1). In its everyday use, the term "data fusion" generally refers to that of Level 1 fusion. This is not the most accurate interpretation of the term. In Level 1 data fusion, all sources of information, including track data and attribute data, are combined to form a composite tracking picture. In order to more fully understand the big picture on sensor fusion, all 5 levels are discussed below, starting with the Data Fusion Functional Model [Ref. 5]:

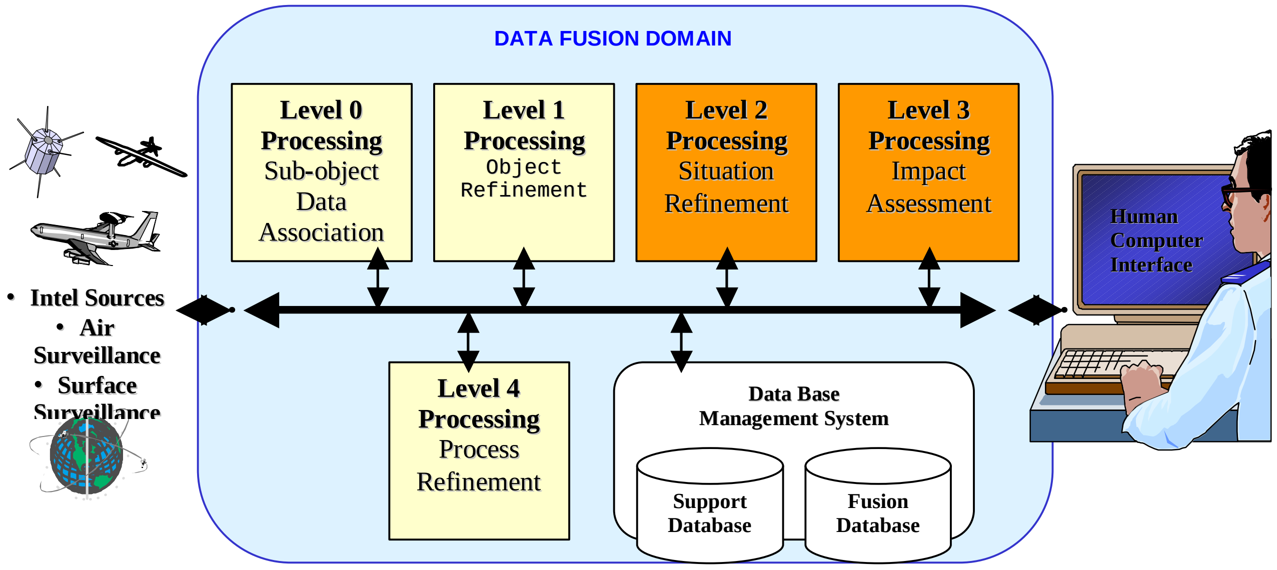
DATA FUSION DOMAIN
Level 0 Processing Sub-object Data Association
Level 1 Processing Object Refinement
Level 2 Processing Situation Refinement
Level 3 Processing Impact Assessment
Level 4 Processing Process Refinement
Data Base Management System
Support Database
Fusion Database
Human Computer Interface
• Intel Sources
• Air Surveillance
• Surface Surveillance

Figure 1. The Data Fusion Functional Model*.

* Reproduced From [Ref. 5]

1. Level 0 - Sub-Object Data Assessment:

Level 0 is considered to be pre-object processing. By definition [Ref. 3], Level 0 fusion is, "estimation and prediction of signal/object observable states on the basis of pixel/signal level data association and characterization." In other words, what kind of targets or signals are expected in the area of interest? From this early assessment, we can estimate and predict the signal or object observable states. Types of data would include signal detected on the basis of integration of a time-series of data or feature extraction from a region in imagery.

2. Level 1 - Object Assessment:

Level 1 fusion is the level that is normally associated with the term "sensor fusion," and is the primary focus of this Thesis. At this level, the raw measurements are used to estimate the current states of each entity. All organic sensor information is combined with off-board information including data links, SATCOM data, Order of Battle databases, ATO/SPINS/ROE, and other database information. During "object assessment," estimation and prediction of object states occurs based on the measurements and/or observations. The kinematics and attributes of the object answer the questions of where and who the object is. According to the JDL [Ref. 3], Level 1 fusion is, "estimation and prediction of entity states on the basis of observation-to-track association, continuous state estimation (kinematics), and discrete state estimation (target type and ID)." Mainly, Level 1 fusion provides the composite tracking picture.

3. Level 2 - Situation Assessment:

In Level 2, a thorough threat assessment is conducted based on the current intelligence reports and other measurements of troop/hardware movements, communications, etc. By JDL definition [Ref. 3], Level 2 fusion is, "estimation and prediction of relations among entities, to include force structure and cross force relations, communications and perceptual influences, physical context, etc." Level 2 fusion involves associating the hypothesized entities or tracks into aggregations. According to the JDL [Ref. 3], the aggregate state can be represented by a network of interconnectivity and relations between the elements. The relations considered include physical, organizational, informational, and perceptual according to the mission of the entity. The term 'situation' is used to describe an aggregate object of estimation.

There are many different approaches to fusing the information at Levels 2 and 3. For example, Bayes theorem and Bayesian belief networks, neural networks, fuzzy logic, genetic algorithms, and Hidden Markov Models (HMM) [Ref. 5] can be used to model the states and estimate situations.

4. Level 3 - Impact Assessment:

The JDL [Ref. 3] definition of Level 3 fusion is, "estimation and prediction of effects on situations of planned or estimated/predicted actions by the participants to include interactions between action plans or multiple players (e.g., assessing susceptibilities and vulnerabilities to estimated/predicted threat actions given one's own planned actions)." The Impact Assessment is also known as Threat Refinement [Ref. 6]. Threat Refinement is used to estimate enemy capabilities, identify threat opportunities, estimate enemy intent, and determine levels of danger to the friendly forces. In doing so, Level 3 fusion focuses on estimating the likelihood of hostile actions, and determining the projected outcomes if hostile actions do occur. Part of Level 3 fusion [Ref. 5] is threat prediction, which attempts to answer the following question: Based on enemy capabilities (firepower and preparedness), what enemy actions pose a threat and to what extent could the enemy damage friendly forces? One key element to predicting the threat is determining the possible enemy courses of action (COA). These COA's could be based on lessons learned form previous actions, or hypothesized based on terrain and natural barriers such as cliffs, lakes, rivers, and oceans. To quantify the COA's, the likelihood of each possible COA is estimated based on the observations.

During the generation of COA's, the information from the lower level fusion engines would be necessary for accurate and sensible predictions. For each course of action, the impact to blue forces must be assessed to determine the probable outcome.

From this evaluation, the most threatening COA's can be identified for the battlefield commander. For example, the Enemy Order of Battle would include all of the enemy forces in the battlespace. Based on the units involved, the hardware, the terrain, and many other observations about the enemy, the candidate COA's can be determined. Depending on their movements (the measurements or observations), the likelihood that the movements are associated with a particular COA can be determined. This likelihood can be determined for each candidate COA, and the unlikely COA's can be discarded. At this point, the possible outcomes of particular enemy actions can be determined. Are the enemy forces assembling, attacking, defending, or deploying? Or, is a particular unit attacking while another is staying back to defend against a counter attack? Next, a determination can be made about the likely outcome of an engagement for each possible COA. From this determination, the threat level to friendly forces can be predicted using a simple decision tree. Finally, an overall likely threat to friendly forces from all possible enemy courses of action can be determined. Should an engagement occur, what is the likely outcome?

5. Level 4 - Process Refinement:

First, a definition is necessary to complete this section. According to JDL [Ref. 3], Level 4 fusion is, "adaptive data acquisition and processing to support mission objectives." In Level 4 fusion, resources are assigned tasks to support the formal relationship between estimation and control, and association and planning. Simply put, Level 4 fusion is resource management to support the particular operator needs. Fusion across the levels takes place to arrive at the most probable enemy course of action based on the measurements, the enemy's capability, and the possible courses of action [Ref. 7]. The operator requirements drive the optimization process of Level 4 fusion. Particular information needs can be used to cue sensors or to optimize the fusion process at each of the other levels.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SENSOR FUSION TECHNIQUES AND PRACTICES

A. TRACKING FUNDAMENTALS

1. General

The fundamental building block of any sensor fusion algorithm is the tracking algorithm associated with each of the sensors on the sensor platform. Issues relevant to tracking are: track initiation, track maintenance, drop track criteria, and handling of false tracks. Additionally, how the raw data is combined within the processors and mission computer can have an impact on how the data is combined at the data fusion level. Primarily, this discussion will focus on the algorithms required for track 'maintenance' vice track initiation or drop track criteria.

2. Sensors and Sources

In an ideal situation, every threat and possible outcome of an engagement could be considered prior to firing a first shot. Naturally, the quality of fusion information given to decision makers is only as good as the sensors and sources of the information that are used as inputs. Sensor types are as diverse as the platforms that carry them. Radars can vary by mission, type, low/high frequency, scan types, pulse repetition frequencies, and by many other differences. Electronic Surveillance Measures systems vary in their abilities to distinguish signal parameters, measure angles of arrival, or determine signal identification. For the purposes of Sensor Fusion, sensors and sources provide the raw and preprocessed data to fusion systems for composite processing. The following definitions are included in Waltz and Llinas [Ref. 2].

a. Sensors are devices that detect and/or measure physical phenomena. Remote Sensing describes a sensor that measures a distinguishing phenomenon that was transmitted through a medium, such as with an ESM system.

b. Sources describe the variety of originators of data, such as observations, intercepted communications or other data, a priori information such as map data (terrain, roads, cities, lakes, rivers, etc.), sea lanes, flight routes, etc., and other archived data such as the OOB/EOB, the ATO/SPINS/ROE, and intelligence data.

c. Links are the communications and connectivity from the sensors and sources to the data fusion processing nodes.

3. Detection and Tracking

Sensors detect the presence of signals and estimate the parameters of those signals. The signal processing of the sensor data results in a measurement for that current sample time [Ref. 1]. Tracking is the result of processing the measurements for the purpose of estimating the current state of the target. The state estimate can consist of the target kinematics and attributes. According to Bar-Shalom [Ref. 1], measurements are the observations of the target, and are usually corrupted by noise from the processing sensor or the signal transmission medium. The characteristics of the measurement are dependent on the sensor type making the measurement. For example, a 3-D radar would directly measure the range, azimuth, and elevation to the target (relative to the sensor). A 2-D radar would measure the range and azimuth to the target. A passive system such as an ESM system would measure the direction of arrival (DOA), but could also determine many of the signal characteristics (such as signal strength, frequency, and pulse repetition frequency). The noise in the measurements is the result of uncertainty in the process, such as detection false alarms, clutter, other targets, and deception/countermeasures [Ref. 1]. For the purposes of this thesis, track initiation and drop criteria are not addressed.

4. Data Association

Any discussion of tracking algorithms would be incomplete without a description of data association methods. While the term 'tracking' is used for describing the state estimation process, 'association' is used to indicate the process of matching the data or knowledge/information to the track [Ref. 1]. The data association process answers the question, "which set of data belong to which track?" According to Waltz and Llinus [Ref. 2], data association is,

> the process of relating individual sensor measurements (data) to other measurements to determine if they have a common source (e.g., target or event). Although the measurements may be referenced to different coordinate systems with different spatial accuracy's or resolutions, the association process must relate each measurement to a number of possible sets of data, each representing a hypothesis to explain the source of the measurement: (1) The false alarm set, indicating that the measurement is unreal and to be ignored, (2) The new target set, indicating that the measurement is real and relates to a target for which there are no previous measurements, (3) An existing set of previous measurements related to a single target. A set exists for each previously detected target.

By Bar-Shalom [Ref. 1], the three categories of data association are measurement-to-measurement association, measurement-to-track association, and track-to-track association. Measurement-to-measurement association is used for track initiation. Measurement-to-track association is used for maintaining the track, such as predicting the current state of the target based on the likelihood that the measurement is actually from the target. Track-to-track association is used in a multiple sensor environment, when two or more sensor systems are reporting well-established tracks. When measurements (or track data) are available from more than one sensor, a process called ‘data registration’ is required to align the sampling times of the sensor data before track-to-track association can be used to combine the data [Ref. 1].

The methods of data association require a likelihood or probability measure to evaluate alternatives [Ref. 1]. For example, a measurement could be ‘associated’ with a current track, another track within the same detection region (gate), a clutter point, or some other anomalous situation. Since it cannot be known with absolute certainty what the measurement is actually associated with, the previous hypotheses or guesses are typically evaluated based on the likelihood that they are the correct guess. Typically, target gates are used to eliminate measurements from consideration in the development of hypotheses. Measurements that fall outside of the gated region are not used in the state estimation of the target. Target ‘gates’ are discussed in the next section. Only the measurements that fall within the gate are considered in estimating the state of the target, depending on the algorithm used. Each measurement can be assigned a likelihood or a probability that it is the measurement that is most correct for updating the state.

Numerous statistical approaches exist for assigning likelihoods or probabilities to the data association process. Several of the methods for data association include the Maximum Likelihood Estimation, Bayesian Statistics, Evidential Reasoning, Dempster's Rule, and Dempster-Shafer Reasoning.

5. Gating

Gating is used to narrow the search around a predicted target position for the next update or measurement. A three dimensional region or volume of probability surrounds the predicted position of the target. According to Blackman and Popoli [Ref. 8], gating is used to eliminate unlikely observation-to-track pairings. If an observation falls within the gated region, the track can be updated with that observation. Measurements or observations that fall outside the gate are not considered in updating the target state at the time of the measurement. Which returns within the gated region will be used to update the track depends on the data association algorithm used. Data association methods will be discussed in the next section. A gated region can take many different forms including rectangular, ellipsoidal, spherical, etc.

Maneuver gating is also an important consideration, depending on the tracking algorithm used. According to Blackman [Ref. 8], a maneuver gate closely models the most severe potential maneuver that a target can perform. Accordingly, the region can be expanded or stretched in any direction depending on the anticipated maneuver.

Three simple approaches to data association within a tracking gate follow:

a. Nearest Neighbor

The Nearest Neighbor (NN) approach is the simplest measurement-to-track association method. With NN, the measurement that is closest to the predicted position of the track is used to update the track state, provided that the measurement is within the tracking gate [Ref. 1].

b. Global Nearest Neighbor (GNN)

The global nearest neighbor (GNN) method is a very simple and widely applied method for data association [Ref. 8]. Under the GNN method, a hypothesis for all the possible associations is made for updating the target state. Only the most likely association is used to update the state and the other possibilities are discarded. Problems occur when more than one measurement occurs in the tracking gate and the formation of a solution occurs with an association assignment matrix.

c. All Neighbors Method

The All Neighbors approach is used in the Probabilistic Data Association Filter, which will be discussed in the section on tracking algorithms.

B. TRACKING ALGORITHMS

1. Batch Processing

Numerous techniques have been used for this simplest case of tracking a single target that is not maneuvering. The approach is to collect a number of hits on the target and batch process the data to produce a track [Ref. 9]. With batch processing, the more hits collected, the better the solution. However, the computational requirements are more demanding as the number of hits increases, making batch processing an impractical solution for most surveillance systems. Every time a new measurement is collected, all the previous measurements are used to calculate the new state, making batch processing very cumbersome.

2. Prediction-Correction Methods

One of the advantages of prediction-correction, or recursive, methods is that the state update depends only on the previous state and the current measurement. According to Hutchins [Ref. 9], the simplest trackers are only designed for straight-line motion (SLM), yet target accelerations can be handled by increasing the gain of the filter or by increasing the noise term of the state equations. The more complicated trackers are designed to compensate for turning motion, or to handle multiple hypotheses of target motion, or to sort through clutter. Tracker types can be combined to allow multiple sensor, multiple target tracking, and data association.

a. Alpha-Beta Tracker (? -?)

The Alpha-Beta tracker is the simplest constant-gain tracking algorithm. The tracker has poor performance, but requires a very low computational load. In this case, a constant gain matrix is set up for the target position update equation. The alpha-beta tracker is used in tracking systems where position measurement updates are available and the state vector consists of positions and velocities [Ref. 9]. The value of the gain is preset for handling straight-line motion or turning motion. When the gain is set to compensate for turning motion in a target, the straight-line performance will suffer somewhat. This is also true for many of the trackers in the discussions that follow. An extension of the alpha-beta algorithm is the alpha-beta-gamma tracker, which includes accelerations in one state vector [Ref. 8].

b. *Constant Gain Kalman Filter (CGKF)*

The Constant Gain Kalman Filter (CGKF) is a simplified case of the Kalman Filter [Ref. 9]. Instead of updating the covariance matrix at every measurement update, the covariance is taken to be constant. Using the assumption that the covariance matrix will approach a steady state value over time, the Algebraic Riccati Equation associated with the linear, time-invariant, discrete time system can be solved numerically. For this case, the MATLAB function *dlqe* (discrete-time linear quadratic estimate) can be used to determine the values of constant covariance and constant Kalman gain. Then the Kalman update equations depend only on the previous state, the measurement, and the constant gain. Although the CGKF is not an optimum solution, it is not heavily burdened computationally.

c. *Kalman Filter (KF)*

The Kalman Filter (KF) is the basis upon which many of the more advanced algorithms are designed. The Kalman filter is an optimum solution to the sequential least squares problem, meaning that the least square error is minimized. It is a sequential algorithm because it only depends on the most current measurement and state estimate, and the associated measurement and state prediction covariance matrices [Ref. 9]. The Kalman filter is not very computationally demanding, but the filter is not designed to handle maneuvering targets, clutter, or multiple targets. The filter can be adapted to handle maneuvering targets, but the solution is no longer optimum. Examples of this phenomenon will be shown in the Results section.

d. Extended Kalman Filter (EKF)

The Extended Kalman Filter (EKF) is used in cases where the mapping of coordinates is non-linear. The EKF is most applicable in cases where the measurement process is non-linear, or the target dynamics are non-linear [Ref. 8]. According to Bar-Shalom [Ref. 1], the nonlinear transformation may introduce bias in the solution, the covariance calculation is not necessarily accurate, and the EKF can diverge if the initial conditions are inaccurate. The EKF was not examined thoroughly in this thesis.

e. Interacting Multiple Models (IMM)

The Interacting Multiple Models tracker is used to predict the current state of the target using two or more different models. For example, if the target is expected to be a maneuvering target, the models used could be a straight-line motion (SLM) model, a left turn model, and a right turn model. Other models used could be a variety of turn rate models or climbing/descending models. The number of models used is dependent on the application. In this thesis, the IMM used is a 2-model IMM, where the two models differ only by the noise term (one for SLM, and one for turning motion).

For the IMM estimator, multiple state equations are used to describe each of the different modes of operation [Ref. 9]. A Markov transition matrix is used to specify the probability that the target is in one of the modes of operation. Usually, these values are chosen heuristically. In the case of the 2-model IMM used in this thesis, the chosen probabilities were (1) A 10% probability that the target would turn if in SLM at the measurement time, and (2) A 33% probability that the target would return to SLM if in a turn at the time of the measurement. Similar to the "soft-switching" discussed in an

Air Force Research Lab report [Ref. 10], the model probabilities are updated at each new measurement, and the resulting weighting factors are used in calculating the state. In other words, no gating decision is required for the tracker to function properly. See Chapter IV for a more thorough description of the algorithm.

f. Multiple Hypothesis Tracker (MHT)

The MHT is a very complex, yet flexible approach to solving the multiple target data association problem [Ref. 9]. Even in a very simple case where one measurement can be associated with one track, numerous possible hypotheses exist as to the precise nature of the association: (1) the measurement may be associated with the currently held track, (2) the measurement may be associated with a new track, (3) the measurement may be associated with no track. With multiple tracks and multiple measurements, the number of hypotheses can grow dramatically. Each hypothesis receives a score, which is dependent on the probability (likelihood function) that the hypothesis is correct [Ref. 9]. All the hypotheses can be maintained from scan to scan, or the hypotheses can be pruned back to the most likely one or two track solutions. For this reason, an MHT is a very computationally demanding algorithm and it requires a great deal of memory.

g. Probabilistic Data Association Filter (PDAF)

The Probabilistic Data Association Filter (PDAF) is used to compensate for clutter that is received as possible valid target returns within a gated region around the predicted target position. The PDAF is a suboptimal Bayesian algorithm that associates probabilistically (using the likelihood function) all the validated measurements to the target of interest [Ref. 1]. For this reason, the PDAF is also known as the “all-neighbors” data association approach. The validated measurements, also known as the neighbors, are combined using a weighted likelihood function in the algorithm to account for measurement uncertainty. The state estimate is updated with all the validated measurements weighted by their likelihoods of having originated from the target, i.e. the combined innovation. To account for the measurement uncertainty, an additional term is added to the covariance update equation. As opposed to the nearest neighbor approach to data association, the PDAF is an ‘all neighbors’ data association approach.

h. Joint Probabilistic Data Association Filter (JPDAF)

The Joint Probabilistic Data Association Filter (JPDAF) is an extension of the PDAF to include multiple targets in clutter [Ref. 1]. The JPDAF allows overlapping validation regions, meaning that more than one target may be present within the gate for each update. This effect causes a persistent interference over several sampling times. One of the simplifying assumptions of the JPDA approach is that the number of targets is known. Also, the false measurements are uniformly distributed across the validation region, meaning the extra computation time is expended evaluating all measurements as if they were valid.

C. ADVANCED TRACKING ALGORITHMS

1. General

In order to handle multiple sensors and multiple targets that may be maneuvering, additional methods are necessary. These methods improve on the basic characteristics of the trackers discussed in the previous section.

2. Multiple Targets

The multiple target tracking problem is discussed extensively in Bar-Shalom [Ref. 1]. If the number of targets is know, the JPDAF is a well established algorithm for tracking the targets. Since the JPDAF is a Bayesian approach, all the possible targets that can be updated by a measurement are considered simultaneously. Additionally, all the validated measurements that fall within the tracking gate are considered in updating the track. However, if the targets are maneuvering, or the number of targets are unknown, some other implementation of the filter is required.

Advanced tracking algorithms are designed to combine the best attributes (or the most practical attributes) of the more basic tracking and data association techniques. For example, combining the IMM with the JPDA allows multiple model tracking with the 'all neighbors' approach to data association. In fact, many of the algorithms can be simplified to allow a sub-optimum solution while preserving the practical aspects of the algorithm. For example, the JPDA could be greatly simplified by forcing nearest neighbor data association (NNJPDA). Combining this new filter with an IMM would result in a filter that could track maneuvering targets with nearest neighbor data association (IMMw/NNJPDAF).

3. Multiple Sensors

In multiple sensor tracking scenarios, the sensors can be of the same type in different locations, or of different types in the same or different locations. A primary consideration in the multiple sensor problem is the time at which the measurements are collected. If the sensors are perfectly synchronized in time, measurement-to-measurement associations can occur, followed by centrally located association and tracking [Ref. 1]. A more common approach occurs where each sensor performs the measurement-to-track association and tracking, and then track-to-track association and fusion occurs between the sensors.

When performing track-to-track association and fusion, a test must be performed to determine if two tracks belong to the same actual target. This is done by comparing the state estimates of the two tracks [Ref. 1]. To test the hypothesis that two tracks belong to the same target, the difference between the state estimates is compared to a threshold value. If the test passes, the fusion of the estimates can be carried out with a simple equation that treats the error covariances of the two sensors independently. The resulting state estimate is usually more accurate than the single sensor case.

D. ATTRIBUTE TRACKING/FUSION

According to Drummond [Ref. 11], the use of the term ‘attributes’ to describe all the characteristics of a target is too broad due to the way that the data is treated for tracking purposes. By Drummond’s definition, *features*, *attributes*, *and categorical features*, are more descriptive terms. A feature is a characteristic obtained from a continuous sample space to include examples like target size, radar cross section, or other signature data. An attribute is based on information obtained in a discrete sample space. Examples of attributes include target type, radar type, number of engines, or the IFF information. The distinction is made between features and attributes because of the way the information is processed. When Bayesian techniques are used, features are processed according to their probability density, and attributes are processed based on their associated discrete probabilities. Categorical features are another type of data that include information that is already known, such as the wingspan of potential target types, or expected IFF information. This information can be measured directly, obtained over time, or combined in a way that allows a comparison to known characteristics of targets, thereby allowing classification into a finite set of categories.

The feature and attribute data is most commonly used for target classification and identification, or for Combat Identification (CID). This is an important concept in Level 1 Sensor Fusion, the detection and estimation of track attributes. However, a distinction is made between these inherent features or qualities of a target and the *behavior* of the target [Ref. 8]. The perceived target behaviors or intentions are the result of higher level sensor fusion, namely the situation assessment (Level 2). According to Blackman [Ref. 8] and Drummond [Ref. 11], the features and attributes of a target can be detected, passed through a thresholding process, estimated, and tracked, similar to tracking based on target kinematics.

E. SENSOR FUSION

The sensor fusion problem is a culmination of all the aforementioned tasks associated with sensor/multiple sensor tracking, attribute tracking, and track-to-track data association. One of the key features of a sensor or data fusion system or algorithm is the necessity to perform data registration. Also, in order to share the information generated in a multiple sensor environment, a robust communications system must be in place to allow timely and accurate sharing of track information.

The sensor registration problem necessitates a correction in track databases due to systems that are not synchronized in time, and corrections due to common reference frame differences (such as gridlock problems). Also, according to Blackman [Ref. 8], many other sources of registration error can contribute to misalignment in track information.

1. Coordinate system errors: misalignment of coordinate axes in the measurement systems.
2. Bias error: errors due to range and bearing bias error.
3. Location error: errors due to variation in navigation solutions.
4. Other error sources: radar refraction/ducting errors, bias errors from polar-to-Cartesian coordinates.

Finally, the architecture of sensor fusion systems are addressed in numerous publications including [Ref. 12], and are not thoroughly discussed in this paper. Whether a fusion system will process data in a centrally located database management system or be distributed to the sensor platforms is an important consideration. In the case of a distributed system, the platform that has the most correct solution about the position, identification, and intentions of a contact may not be readily known. Indeed, the confidence in the data needs to be addressed for both types of system architectures.

The information received may be vital to the mission. A Battle Group Commander may want to maximize the situational awareness of his platforms via a common operational picture. An aircraft responding to a time-critical strike request will need maximum information about target location and movement, target type or identification, and possible threats to his platform.

IV. FUSION SIMULATOR

A. *FUSIM* OBJECTIVE

The objective of the *fusim* program is to provide acquisition managers a tool for evaluating tracking and sensor fusion algorithms. Also, this first edition of the program provided a basis of study for this Thesis. The fusion simulator was written in MATLAB 6.0 for the following reasons: (1) The MATLAB software is widely accessible and runs on any personal computer. (2) MATLAB is compatible with other languages including C++ and Java. (3) MATLAB is highly flexible and optimized for vector/matrix operations. (4) MATLAB can be programmed to utilize multi-dimensional arrays. (5) MATLAB can be used in an object-oriented programming environment.

The fusion simulator (FUSIM) was written from the perspective of the sensor platform and the operator. Any type of platform can be selected for the sensor platform, either airborne or surface, maneuvering or non-maneuvering, or a fixed site. FUSIM is very flexible for selection of targets. The user can select any number of high-speed/low-speed/fixed maneuvering or non-maneuvering targets for maximum flexibility in a 2-dimensional simulation environment. Also, the user can select up to 4 different sensors for the sensor platform. For output, the user can select a real-time output plot (PPI) or the PPI and the sensor/target/tracker Least Square Error plots.

B. *FUSIM* DESCRIPTION

The *fusim* program was written to give a user flexibility in selecting sensor platforms, up to four sensors associated with that platform, the target types, the problem orientation, and the tracking algorithms to be used with the sensors. Table 1 is a list of potential *fusim* configurations that a user could select.

Sensor Platforms	Airborne	Maneuvering		Surface	Moving – Maneuvering	
		Non-Maneuvering			Moving – Non-Maneuvering	
		Fixed (like an aerostat radar)			Fixed Site	
Potential Sensors	Airborne Surveillance Radar	Airborne Fighter Radar	Non-Cooperative Target Radar	IFF Interrogate	Airborne ESM error model	Infrared Search and Track error Model
	Surface Surveillance Radar	Surface Fire Control Radar	Eastern IFF	IFF Reply	Surface ESM error model	Ownship Navigation model
Target Types	Airborne	Maneuvering		Surface	Moving – Maneuvering	
		Non-Maneuvering			Moving – Non-Maneuvering	
		Fixed			Fixed Site	
Tracking Algorithms	Probabilistic Data Association Filter (PDAF)	Interacting Multiple Models (2-model) Filter (IMM)		Kalman Filter (KF)	Constant Gain Kalman Filter (CGKF)	

Table 1. List of Potential Configurations for the *fusim* Program.

Within the *fusim* program, the FUSIM file is the main control file for the entire simulation. FUSIM calls the various associated functions to set up the needed matrices and variables, creates the truth data for the targets, adds the measurement error, sends the noisy measurements to the trackers, and then stores and plots the target states and associated least square errors. The FUSIM function is called from the MATLAB command line. A complete description of the *fusim* program and the related functions is included in Appendix A. The MATLAB code for all the functions is included in Appendix B.

C. THE MATHEMATICS OF *FUSIM*

The *fusim* program runs on a few simple concepts of modeling and simulation. For example, the program utilizes synchronous timing for the sensors associated with the sensor platform, making the data association problem a little easier to deal with. This section deals with the mathematics of the *fusim* program, describing the construction of the various matrices and vectors, and describing their interactions at specific points in the program.

1. Startup

Fusim begins with the START function for initializing a variety of variables including the Extraction Matrix (H) and the Discrete Time State Equation Transition Matrix (F). Also, all the target matrices are initialized and the simulation timing is determined.

<u>Extraction Matrix (H)</u>

The Extraction Matrix simply extracts the x,y components from the 4 x 1 state vector. As used in this program, H is a 2x4.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}; \qquad state = \begin{bmatrix} x \\ vx \\ y \\ vy \end{bmatrix} = \begin{bmatrix} x \text{ position} \\ x \text{ velocity} \\ y \text{ positon} \\ y \text{ velocity} \end{bmatrix}$$

Total Simulation Time (t)

The total simulation time in minutes is input by the user via the START.MAT file. After reading in the value, the simulation time is converted to seconds.

$$simstart = [simulationtime, nloops, deltatime, DLRP, N/A, filterselection1-4]$$

$$t = simulationtime * 60$$

Simulation Time (simt)

The cumulative simulation time is stored in the (simt) variable. The value of (simt) starts at zero and runs up to the total number of seconds in the simulation.

Number of Simulation Loops (nloops)

The value of (nloops) is usually 1 for a simulation with a real-time output. For multiple runs, a value of 20 provides a good indication of performance. For sufficient averaging of the Monte Carlo runs, a value of 100 or 200 is required. However, the computation time is dramatically increased.

Sampling Time (delta)

The sampling time is used to determine the time step and the total number of measurements based on the simulation time.

$$delta = deltatime * 60$$

Number of Time Steps (nsamples)

The number of time steps is determined using the value of delta and the total simulation time. This value is used by the main program loop for the total number of measurements.

$$nsamples ? round(t / delta)$$

Transition Matrix (F)

The Transition Matrix is used to iterate target motion through time based on the time delta. The F-matrix is multiplied by the applicable state matrix or truth state to determine the next target position [Ref. 9].

$$F ? \begin{bmatrix} 1 & delta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & delta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Target Matrices (A) and (A1)

The Target Matrices result from reading the TARDAT.MAT file and separating the targets into maneuvering (A) and non-maneuvering (A1) targets. This separation is accomplished by performing a simple test: If the turn time in the last position of each target vector is a zero, the target is non-maneuvering regardless of the leg time input. The sensor platform and all targets (maneuvering and non-maneuvering) have the same format. The targets have emitters instead of sensors. For all non-maneuvering targets, the (F) matrix is used to transition the state vector. For all maneuvering targets, the (Fturn) transition matrix is used while the target is maneuvering.

$$A = [N/S, y = pos, E/W, x = pos, heading, speed, altitude, type, name$$

$$sensor1, sensor2, sensor3, sensor4, comm\#, legtime, L/Rturns, turntime]$$

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 270 & 175 & 25000 & 1 & 1 & 1 & 7 & 9 & 11 & 0 & 10 & 1 & .43 \\ 0 & 400000 & 0 & 300000 & 170 & 350 & 25000 & 1 & 1 & 1 & 5 & 6 & 8 & 10 & 9 & 1 & .45 \\ 0 & 500000 & 0 & 200000 & 240 & 400 & 20000 & 1 & 1 & 1 & 5 & 6 & 8 & 10 & 7 & 0 & .5 \\ 0 & 200000 & 0 & 400000 & 330 & 100 & 5000 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 0 & .2 \end{bmatrix}$$

$$A1 = \begin{bmatrix} 0 & 250000 & 0 & 500000 & 070 & 500 & 30000 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 0 & 0 \\ 0 & 60000 & 0 & 60000 & 355 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 700000 & 0 & 100000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

<u>Sensor Platform Position (Sp) and (Spi)</u>

The sensor platform position is initialized with (Spi) by extracting the x-y position, the heading and the velocity from the first target in the TARDAT.MAT file. The sensor platform is always the first row vector of the TARDAT.MAT file. The sensor position used in all calculations of target motion is based on the navigation solution of the sensor platform (Sp). The sensor platform is sometimes referred to as Ownship.

<u>Ownship Sensors (ownsens)</u>

The sensors associated with the sensor platform (ownship) are determined based on user preference in the TARDAT.MAT file. Vector positions 10 – 13 contain the preferences for sensors as defined in the SENSERROR function.

$ownsens = [1 \quad 7 \quad 9 \quad 11]$ (See Appendix A for a complete description of the sensors)

Maneuvering Target Transition Matrix (Fturn) and (ownfturn)

The (Fturn) matrix is used to iterate target motion through time whenever the target or sensor platform is in a maneuvering condition. The matrix is set up by the FTURN function, which determines if the target is turning left or right and assigns the turn g's. The (Fturn) matrix is based on each target's heading, speed, and turn g values. For this simulation, the (Fturn) matrices are stacked for each target resulting in a 4*(# targets) x 4 matrix. The (Fturn) matrix is used only for generation of truth data.

$$? \;=\; \frac{\|a\|}{\|v\|} \quad \text{(turn rate in radians/sec);} \;\; ? \;?\; \mathit{delta}$$

$$\text{fturn} \;=\; L^{-1}\{(s\mathbf{I} - \mathbf{A})\} = \; e^{A?} \;=\; \begin{bmatrix} 1 & \frac{\sin(?\;?)}{?} & 0 & \frac{1?\cos(?\;?)}{?} \\ 0 & \cos(?\;?) & 0 & ?\sin(?\;?) \\ 0 & \frac{1?\cos(?\;?)}{?} & 1 & \frac{\sin(?\;?)}{?} \\ 0 & \sin(?\;?) & 0 & \cos(?\;?) \end{bmatrix}$$

where A is the System Matrix [Ref. 1, 9].

Sensor Numbers (nsens)

The (nsens) vector holds the values corresponding to the desired sensors of the sensor platform. The variable (ns) holds the total number of sensors associated with the sensor platform. The number of sensors can vary from 1 to 4. When more than one sensor is associated with the sensor platform, many of the vectors and matrices in the simulation are given a third dimension to allow analysis of the additional sensors.

nsens ? ?1 7 9? (See Appendix A for a complete description of the sensors)

ns = 3 (in this case)

Tracker (trkr)

The (trkr) vector is returned from the SETUPTRACKERS function containing the tracker that corresponds to each sensor. If the primary tracker is set to 1 by the user, the sensor1/tracker1 pair is set to 1. The 1 corresponds to the Probabilistic Data Association Filter. Subsequent pairs are automatically set by the SETUPTRACKERS function.

trkr = [sensor1/tracker1 pair, sensor2/tracker2 pair, sensor3/tracker3 pair,...]

$trkr\ ?\ ?2 \quad 3 \quad 4 \quad 1?;$

In this case, sensor 1 corresponds to tracker 2 (IMM), sensor 2 corresponds to tracker 3 (Kalman), sensor 3 corresponds to tracker 4 (Const Gain Kalman), and sensor 4 corresponds to tracker 1 (PDA).

2. Initialization

Initial x (xin) and (xim)

The x vectors (xin) and (xim) are based on the initial values input by the user as the initial position, velocity and heading of the target. To keep them separate, xin contains the initial values for the non-maneuvering targets, and xim contains the maneuvering target data. The vectors contain the initial true states stacked for each target. The x2n and x2m vectors are simply the next truth data point generated by the TIMESTEP function. The TIMESTEP function simply multiplies the (F) matrix by the 4 x 1 truth state vector.

$$\text{xin} = \begin{bmatrix} x \\ vx \\ y \\ vy \end{bmatrix}; \quad \text{xim} = \begin{bmatrix} x \\ vx \\ y \\ vy \end{bmatrix}; \quad \text{x2n} = \text{F*xin} = \begin{bmatrix} x2 \\ vx2 \\ y2 \\ vy2 \end{bmatrix}$$

Maneuver Time Matrix (mantime)

The Maneuver Time Matrix (mantime) is set up by the MANMATRIX function and contains a column vector of times for each maneuvering target. The vector times are based on the leg times and turn times as defined by the user in the TARDAT.MAT file. For each target, the column vector holds the cumulative leg time and turn times. During matrix construction, if the maximum time is greater than the total simulation time, the last element is set to the maximum time. The size of the matrix is kept consistent by filling in the shorter vectors with zeros.

$$mantime = \begin{bmatrix} 480 & 600 & 400 & 480 & 300 & 600 \\ 580 & 750 & 500 & 540 & 450 & 700 \\ 900 & 900 & 900 & 900 & 750 & 900 \\ 0 & 0 & 0 & 0 & 900 & 0 \end{bmatrix}$$

Initial State Estimation (xhat1)

The initial state estimation for each target is determined by the INIT function and returned with the applicable sensor covariance (R) and prediction covariance (P). Also, the measurement error (derror) is returned for plotting and comparison purposes. The additive plant noise term, Q is also returned. The INIT function is called for each sensor.

$$Q ? \begin{bmatrix} \frac{?^3}{3} & \frac{?^2}{2} & 0 & 0 \\ \frac{?^2}{2} & ? & 0 & 0 \\ 0 & 0 & \frac{?^3}{3} & \frac{?^2}{2} \\ 0 & 0 & \frac{?^2}{2} & ? \end{bmatrix}; \qquad D ? \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{?} & 0 & ?\frac{1}{?} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{1}{?} & 0 & ?\frac{1}{?} \end{bmatrix};$$

The function POLE2CART is called within INIT for each target to obtain the noisy measurement (zcart), the sensor covariance (R), and the measurement distance error (disterror). Using the second set of truth data, the POLE2CART function is called again for use in Lease Squares Initialization. To simplify initialization requirements, the least squares initialization is used for all trackers in the simulation, regardless of tracker type. The INIT function simply calls POLE2CART for each target using the appropriate truth position (xin/m), sensor error values (sigr, sigb, Sv) and sensor position (Sp). Within POLE2CART, the sensor position (sp) is subtracted from the target true x-y (ztrue). Next, the actual range and bearing (r,b) are calculated to the target and the measurement noise is added. The values are converted back to Cartesian coordinates and the sensor position is added back in to get the plotted position of the target. Finally, the measurement covariance in Cartesian coordinates is calculated and the distance error is calculated for determining the measurement error.

$$z \; ? \; ztrue \; ? \; sp; \qquad r \; ? \; \sqrt{z(1)^2 \; ? \; z(2)^2}$$

$$b \; ? \; \tan^{?1}\left(\frac{z(2)}{z(1)}\right)$$

*r = r + sigr*randn*

*b = b + sigb*randn*

$$z = \begin{bmatrix} r\cos(b) \\ r\sin(b) \end{bmatrix} + sp; \qquad fx = \begin{bmatrix} \cos(b) & -r\sin(b) \\ \sin(b) & r\cos(b) \end{bmatrix}$$

*R = fx*Sv*fx';* *ztilde = ztrue – z*

$$disterror = \sqrt{ztilde' * ztilde}$$

Slight biases are introduced with this type of measurement processing [Ref. 1]. These errors are ignored in this simulation. Once 2 noisy measurements are obtained, the Kalman initialization can commence to create xhat and the covariance term:

$$yflipped = \begin{bmatrix} z(2) \\ z(1) \end{bmatrix} = \begin{bmatrix} x2 \\ y2 \\ x1 \\ y1 \end{bmatrix}$$

$$\text{cov} = \begin{bmatrix} R2 & zeros(2,2) \\ zeros(2,2) & R1 + H * F^{-1} * Q * (F^{-1})' * H' \end{bmatrix}$$

*P = D*cov*D'*

*xhat =D*yflipped*

Sensor Platform Initial State (Spst)

The initial state of the sensor platform is determined with the SPINIT function similar to the initializations of the targets. The sensor position state is determined using the same methodology as xhat for the targets.

3. Truth Data

Non-Maneuvering Target Time Step

If the target is not maneuvering at any time during the simulation, generation of the truth data is a simple step:

*xnew = F*xold*

Maneuvering Target Time Step

For maneuvering targets, the truth data generation is a little more complicated. First, a test must be applied to see if each target is in a maneuver condition. This is where the previously described (mantime) matrix is used. If the target is in a maneuver state at the current simulation time:

Target 5

$$mantime = \begin{bmatrix} 480 & 600 & 400 & 480 & 300 & 600 \\ 580 & 750 & 500 & 540 & 450 & 700 \\ 900 & 900 & 900 & 900 & 750 & 900 \\ 0 & 0 & 0 & 0 & 900 & 0 \end{bmatrix} \longleftarrow \text{Turning Condition}$$

simt = 400

*xnew = fturn*xold*

When not in a turning condition, the previous expression is used for straight line motion.

Sensor Platform Time Step

The Sensor Platform is treated separately from the targets. To generate the truth data, the previous truth data is sent to the SPITMEAS function along with maneuver times and the ownship turning motion matrix (ownfturn).

spxtrue = *F*spx2*;   or, *spxtrue* = *ownfturn*spx2* (if maneuvering)

4. Measurement

Once all the truth data is generated, the measurement phase can commence. The truth data is sent to the POLE2CART function for each sensor and for each target to create the noisy measurements. The POLE2CART function calculates the measurements exactly the same as the measurements in the initialization section.

<u>Measurement Truth Data (zre)</u>

The measurement truth data is simply the ztrue vector reshaped into a stack of x-y row vectors. This reshaping is done to make the data more convenient to use with POLE2CART.

$$zre = \begin{bmatrix} x1 & y1 \\ x2 & y2 \\ x3 & y3 \\ \cdot & \cdot \\ . & . \\ xn & yn \end{bmatrix}$$

<u>Sensor Platform Position Update (Sp)</u>

One of the key elements in the POLE2CART function is the sensor platform position because all errors are measured with respect to the position of the sensor. To simulate the sensor platform navigation solution, the truth data is treated the same as the targets. First, the truth data is sent to POLE2CART with some r, ? errors to create the measurement noise (in the SPITMEAS function). This simulates errors in the platform's GPS/Inertial navigation system. Next, the noisy measurement is sent to a Kalman Filter tracking function to create the navigation state update.

Sensor Position State (spstate)

The sensor position state is updated by calling the KALMAN1 function with the previous state (Spst), the prediction covariance (spP), the measurement covariance (spR), the noise term (spQ), and the current noisy measurement (spmeas). The state prediction and update equations follow:

Kalman Prediction:

$Pnext = F*spP*F' + spQ$

$xnext = F*xprevious$

Kalman Update:

$K = spP*H'*(H*spP*H' + spR)^{-1}$

$spP = (I - K*H)*Pnext*(I-K*H)' + K*spR*K'$

$spstate = xnext + K*(spmeas - H*xnext)$

$Sp = H*spstate$

The (Sp) variable is the estimated x-y position as extracted from the state vector.

Noisy Target Measurements (Z1)

For each sensor (with associated errors), a call to the MEAS function results in a vector of noisy measurements for each target. The MEAS function simply calls POLE2CART for each target using the appropriate truth position (zre), sensor error values (sigr, sigb, Sv) and sensor position (Sp). Within POLE2CART, the sensor position is subtracted from the target true x-y (ztrue). Next, the actual range and bearing (r,b) are calculated to the target and the measurement noise is added. The values are converted back to Cartesian coordinates and the sensor position is added back in to get the plotted position of the target. Finally, the measurement covariance in Cartesian coordinates is calculated and the distance error is calculated for determining the measurement error.

For each target, the measurements, measurement covariance, and distance errors are stored (zret, R, derror) and returned to the for further storage and manipulation.

$$z1 ? \begin{bmatrix} x1 & x2 & x3 & . & . & . & x_n \\ y1 & y2 & y3 & . & . & . & y_n \end{bmatrix}$$

$$R ? \begin{bmatrix} R1_{2x2} \\ R2_{2x2} \\ R3_{2x2} \\ \cdot \\ . \\ Rn_{2x2} \end{bmatrix}$$

5. Tracking

The next step in the program is the tracking section. Four different trackers are included in the current configuration of the *fusim* program. A complete listing of the MATLAB code for all the algorithms is contained in Appendix B. The TRACKER function is called for each sensor with the (trkr) vector for determining which tracker is associated with which sensor. The TRACKER function makes the call to the appropriate tracking function. The tracking algorithms used in *fusim* are listed in order of precedence starting with the Probabalistic Data Association Filter.

Probabilistic Data Association Filter (PDAF)

The PDAF [b-s] is the only tracking code included with *fusim* that allows analysis of targets moving in clutter. This implementation of the PDAF is not designed for use with multiple targets or maneuvering targets. However, increasing the additive noise term (Qp) allows sufficient tracking through turns for this analysis.

Initially, the probability of detection (Pd) and the gate density probability (Pg) are set. The gate probability is usually around 1 [Ref. 1]. The number of targets and clutter points (m) are also predetermined. For this PDAF, one target is always assumed to be present. Another simplification for this filter is the calculation of the tracking gate. For each clutter point within the gate, a simple call to POLE2CART is used to generate the point. This assures that the clutter point always falls within the gate. These values do not change in this adaptation of the Probabilistic Data Association Filter.

Pd = 1.0; *Pg = .99;* *m = 2*

For the gate calculation:

$?_{range} \ ?\ 900\, feet$

$?_{bearing} \ ?\ 2 \deg$

$$gate\ \mathrm{var}iance\ ?\ \begin{matrix} ?? & ?^{2}_{range} & 0 & ? \\ ? & & & ? \\ ? & 0 & ?^{2}_{bearing} & ? \end{matrix}$$

qsquared = 1000

*Qp = qsquared*Q*

The weighted predictions:

$\hat{x}_1 \; ? \; F\hat{x}$

$Pnext \; ? \; FPpF^{?} ? \; Qp$

For calculation of the gate volume factor, the semi-major and semi-minor axis in feet:

minor axis, a = 600;

major axis, b = 1500;

$Vk \; ? \; 2?ab$

$? \; ? \; \frac{m}{Vk}$

$\Pr eS \; ? \; HPnextP^{?}$

Calculate the innovation covariance (Sk) for the actual target, based on the measurement:

$Sk \; ? \; \Pr eS \; ? \; R$

$K \; ? \; PnextH ?Sk^{?1}$ Gain

$b \; ? \; ? \; ?|2?Sk|^{\frac{1}{2}} ?? \frac{1 ? PdPg}{Pd} ?$

sum = b

$? \; ? \; zret \; ? \; H\hat{x}_1$

$e1 \; ? \; e^{?\frac{1}{2}? ?Sk^{?1}? ?}$

$sum \; ? \; sum \; ? \; e1$

For each clutter point, the POLE2CART function is called to create the noisy measurements and the association probabilities (?) are calculated.

$?_j \; ? \; zret \; ? \; H\hat{x}_1$

$$e_j ? e^{?\frac{1}{2}? ?Sk^{?1}? _j ?}$$

$$sum ? sum ? e_j$$

$$?_j ? \frac{e_j}{sum}$$

Then, for the combined innovation calculation:

$$?_k ? ?_{i?1}^{m} ?_i ?_i$$

For determining the spread of the innovations term, the calculations are broken down as follows:

For each clutter point (and target),

$$spread ? ? ?_i ?_i ?_i ?? ?_k ?_k$$

$$\tilde{P} ? K ?spread ?K?$$

$$Pc ? Pnext ? KSkK?$$

$$?_0 ? \frac{b}{sum}$$

$$Pp ? ?_0 Pnext ? ?1 ? ?_0 ?Pc ? \tilde{P}$$

Final state estimation:

$$\hat{x} ? \hat{x}_1 ? K?_k$$

Interacting Multiple Models

The 2-model Interacting Multiple Models filter allows tracking of a maneuvering target based on a straight line motion model and a turning model. Initially, the probability of conducting a turn if in straight line motion (alpha) and the probability of stopping a turn (beta) are set. This allows setting up the Markov 2 x 2 ? matrix.

$$? \; ? \begin{bmatrix} 1 ? ? & ? \\ ? & 1 ? ? \end{bmatrix}$$

In this tracker, ? = 0.1 and ? = 0.33333. These values can be changed for user preference.

Initial State Likelihood: (straight line track)

$? \; ? \begin{bmatrix} 1 \\ 0 \end{bmatrix}$; Initially, $?_1$ = 1.0, indicating the target is assumed to be in straight line motion (SLM). The state likelihoods are recalculated each time the tracker is called for a new update.

Set the noise terms for the two different models:

*Qi = 1.0*Q;*

*Qturn = 10000*Q;*

Pre-process the cbars and mu's:

$$\bar{c}_1 \; ? \; ?_{11} ?_1 \; ? \; ?_{21} ?_2$$

$$\bar{c}_2 \; ? \; ?_{12} ?_1 \; ? \; ?_{22} ?_2$$

The initial xhat guesses are based on the probabilities that the target will continue straight line if in SLM, turn if SLM, continue turning if in turning motion, or return to SLM if turning:

$$\hat{x}_{z1} \; ? \; \hat{x}_1 \begin{bmatrix} \frac{?_{11} ?_1}{\bar{c}_1} \end{bmatrix} ? \; \hat{x}_2 \begin{bmatrix} \frac{?_{21} ?_2}{\bar{c}_1} \end{bmatrix}$$

$$\hat{x}_{z2} \; ? \; \hat{x}_1 ? \frac{? \, ?_{12} ?_1 }{\bar{c}_2} ? ? \; \hat{x}_2 ? \frac{? \, ?_{22} ?_2 }{\bar{c}_2} ?$$

$$\bar{?}_{11} \; ? \; \frac{?_{11} ?_1}{\bar{c}_1}; \qquad \bar{?}_{21} \; ? \; \frac{?_{21} ?_2}{\bar{c}_1}$$

$$\bar{?}_{12} \; ? \; \frac{?_{12} ?_1}{\bar{c}_2}; \qquad \bar{?}_{22} \; ? \; \frac{?_{22} ?_2}{\bar{c}_2}$$

$$\tilde{x}_{11} \; ? \; \hat{x}_1 \; ? \; \hat{x}_{z1} \; ; \qquad \tilde{x}_{21} \; ? \; \hat{x}_2 \; ? \; \hat{x}_{z1}$$

$$\tilde{x}_{12} \; ? \; \hat{x}_1 \; ? \; \hat{x}_{z2} \; ; \qquad \tilde{x}_{22} \; ? \; \hat{x}_2 \; ? \; \hat{x}_{z2}$$

$$P_{01} \; ? \; \bar{?}_{11} ?P ? \tilde{x}_{11}\tilde{x}_{11}^{?} ?? \bar{?}_{21} ?Pturn ? \tilde{x}_{21}\tilde{x}_{21}^{?} ?; \quad P_{02} \; ? \; \bar{?}_{12} ?P ? \tilde{x}_{12}\tilde{x}_{12}^{?} ?? \bar{?}_{22} ?Pturn ? \tilde{x}_{22}\tilde{x}_{22}^{?} ?$$

Next, the weighted predictions are performed:

$$\hat{x}_1 \; ? \; F\hat{x}_{z1}; \qquad \hat{x}_2 \; ? \; F\hat{x}_{z2}$$

$$P_1 \; ? \; FP_{01}F ?? \; Qi$$

$$P_2 \; ? \; FP_{02}F ?? \; Qturn$$

Before proceeding to the next step, the straight line Kalman gain, turning gain, and covariances are determined separately:

$$K \; ? \; P_1H ??HP_1H ?? \; R?^{?1}$$

$$Kturn \; ? \; P_2H ??HP_2H ?? \; R?^{?1}$$

$$Pi \; ? \; ?I \; ? \; KH ?P_1 ?I \; ? \; KH ?^{?} \; ? \; KRK ?$$

$$Pturn \; ? \; ?I \; ? \; KturnH ?P_2 ?I \; ? \; KturnH ?^{?} \; ? \; KturnRKturn ?$$

$$S_1 \; ? \; HP_1H ?? \; R$$

$$S_2 \; ? \; HP_2H ?? \; R$$

The measurement update step is conducted by calculating the state estimate for straight line and turning, and then combining the results for plotting:

Straight line

$\tilde{z}_1 \ ? \ zret \ ? \ H\hat{x}_1$

$\hat{x}_{slm} \ ? \ \hat{x}_1 \ ? \ K\tilde{z}_1$

Turning

$\tilde{z}_2 \ ? \ zret \ ? \ H\hat{x}_2$

$\hat{x}_{turn} \ ? \ \hat{x}_2 \ ? \ Kturn\tilde{z}_2$

Combining the result for plotting and error analysis, the state estimate:

$\hat{x} \ ? \ ?_1\hat{x}_{slm} \ ? \ ?_2\hat{x}_{turn}$

Score the results (simplified with only two models):

$$?_1 \ ? \ \frac{e^{?\,?\,?\,?\frac{\tilde{z}_1 ?S1^{?1}\tilde{z}_1}{2}??}}{2?\,\|S1\|^{\frac{1}{2}}}; \qquad ?_2 \ ? \ \frac{e^{?\,?\,?\,?\frac{\tilde{z}_2 ?S2^{?1}\tilde{z}_2}{2}??}}{2?\,\|S2\|^{\frac{1}{2}}}$$

$$c \ ? \ ?_1\bar{c}_1 \ ? \ ?_2\bar{c}_2$$

$$?_1 \ ? \ \frac{?_1\bar{c}_1}{c}; \qquad ?_2 \ ? \ \frac{?_2\bar{c}_2}{c}; \qquad ? \ ? \ \begin{matrix} ??_1? \\ ? \quad ? \\ ??_2? \end{matrix}$$

For the next prediction/update:

$\hat{x}_1 \ ? \ \hat{x}_{slm}$; $\qquad \hat{x}_2 \ ? \ \hat{x}_{turn}$

Kalman Filter

The Kalman filter used for tracking the targets is exactly the same code as the equations used in the sensor platform navigation solution.

Kalman Prediction:

$$Pnext = F*spP*F' + spQ$$

$$xnext = F*xprevious$$

Kalman Update:

$$K = spP*H'*(H*spP*H' + spR)^{-1}$$

$$spP = (I - K*H)*Pnext*(I-K*H)' + K*spR*K'$$

$$spstate = xnext + K*(spmeas - H*xnext)$$

$$Sp = H*spstate$$

Constant Gain Kalman Filter

The constant gain Kalman filter is the simplest case included in the *fusim* program.

qsquared = 1000.0;

*Q = qsquared*Q;*

First, the constant Kalman gain is determined using the MATLAB function DLQE for each target being tracked. This value of $\overline{K}$ is used for all subsequent state calculations.

$$\hat{x}_{pred} \ ? \ F\hat{x}$$

$$\hat{x} \ ? \ \hat{x}_{pred} \ ? \ \overline{K} \, ? zret \ ? \ H\hat{x}_{pred} \, ?$$

6. Data Link Simulation

For the data link simulation, bias error and noise errors are added to the state of the primary sensor on the sensor platform. The errors are added to simulate reference frame differences, noise in another platform, and transmission errors. The link report is then formatted to allow many different types of data to be sent as part of the report, including attribute data, and the time of the report. For each target:

*x = x + randn*50 + 100 feet*

*y = y + randn*50 + 100 feet*

*vx = vx + randn*10 feet/sec*

*vy = vy + randn*10 feet/sec*

linkvector = [x,vx,y,vy,hdg,time,......]

The link reports are meant to be used in a track-to-track data association algorithm, which is not available in this first edition of *fusim*. The x and y data are the only data used in this simulation for plotting purposes.

7. Storage and Plotting

For every instance of target motion, the truth data (posout), "measured" position (zout), tracker state (state), and least square error (Lserror) are stored in a matrix. The same data is collected for the sensor platform. After each Monte Carlo run, the current data are added to the previous cumulative data for eventual averaging over the number of runs. Three different plotting functions are provided for plotting the real-time results or for plotting the tracks and the errors for each target and sensor.

D. EVALUATION PARAMETERS

1. The Baseline Target Set

The target set used for all simulations are presented in Table 2. This setup was designed to be flexible enough to add or remove maneuvering or non-maneuvering targets via the TARDAT.MAT file. The format of the TARDAT.MAT file must be followed precisely, with zeros entered where no data is desired.

Track	**Initial Position (x,y)**	**Initial Course (degrees)**	**Initial Speed (knots)**	**Maneuvering? (Y/N)**	**Notes**
Ownship	0000/0000	270	175	Y	Airborne surveillance platform
Target 1	400000/300000	170	350	Y	Maneuvering Fighter Aircraft
Target 2	500000/200000	240	400	Y	Maneuvering Fighter Aircraft
Target 3	200000/400000	330	100	Y	Slow-Moving Air Track
Target 4	250000/500000	070	500	N	Air Liner, Non-Maneuvering
Target 5	60000/60000	355	12	N	Ship
Target 6	700000/100000	0	0	N	Fixed Target/Emitter

Table 2. The target set used for all simulations.

2. Simulation Set

For this initial edition of *fusim*, several different sets of runs were performed to examine the validity of the code, to check the simulation in a multiple sensor/multiple target scenario, and to observe algorithm behaviors in a simple combined solution. The three major tests are described as follows:

<u>Test 1: Track algorithm validity</u> – The validity of the simulation was checked by running single sensor/multiple track test sets and comparing the mean measurement noise to the tracking solution for each track. Initially, zero error was input to check for

problems in the code. For the next set of runs, the noise term (q^2) was adjusted in two of the trackers to compare straight line motion (SLM) and maneuvering target tracking characteristics. Finally, the track solutions were compared by running three different sets of runs using a single sensor with all four trackers. Table 3 shows a breakdown of the tests and the test subsets.

Test 2: Multiple Sensor/Multiple Target Tracking – Test 2 was run to examine the use of multiple sensors and tracker types in a multiple target tracking scenario. Representative plots were produced to show the differences in mean measurement noise and tracking solutions when using four different sensors and four different tracking algorithms. This test was simply a demonstration of the multiple sensor/multiple target case and a precursor to the next test.

Test 3: Simple Combined Solution – A third group of test sets were run to demonstrate a simple combined tracking solution using the multiple sensor/multiple target tracking results of Test 2. The combined state result for each track was simply a weighted average of each of the individual tracker states. The weights were selected empirically based on the tracker performance results observed in Test 1.

The combined state estimate was based on the following weights (percentages) for each of the sensors:

s1:	Sensor 1 = 0.3	(Surveillance Radar)
s2:	Sensor 2 = 0.4	(IFF System)
s3:	Sensor3 = 0.1	(ESM System)
s4:	Sensor 4 = 0.2	(IRST System)

Then, for the combined state estimate:

xhat = s1*xhat(sensor1) + s2*xhat(sensor2) + s3*xhat(sensor3) + s4*xhat(sensor4).

For all of the tests, the runs were conducted with the same target set and target initial parameters (Table 2). Table 3 is a description of the evaluation sets. The sensor descriptions and associated errors are included in Table 4.

TEST	NAME	TEST SUBSET
Test 1	**Track Algorithm Validity**	Set (a) Zero sensor error Set (b) Analyze algorithm noise terms Set (c) Airborne surveillance radar Set (d) Surface fire control radar Set (e) Infrared search and track system simulator
Test 2	**Multiple Sensor Multiple Track**	Set (a) Four different sensors and trackers
Test 3	**Observations on Algorithm Behavior in a Simple Combined Solution**	Set (a) Single sensor type/combined solution with four tracking solutions Set (b) Four sensors/combined solution Set (c) Four sensors/combined solution, modified noise terms for turn performance

Table 3. *Fusim* Evaluation Set.

Sensor	**Range Error**	**Bearing Error**	**Notes**
Airborne Surveillance Radar	100 ft	.005 rad	Simplified model of an airborne 2D radar.
IFF Detection System	50 ft	.001 rad	Simple interrogation and reply range and bearing only.
ESM System	500 ft	.01 rad	A simplified model of an ESM system.
IR Search and Track	6000 ft	.0001 rad	A simplified model of an IRST system.
Surface Fire Control Radar	10 ft	.0001 rad	A simplified model of a Fire Control Radar system.

Table 4. Test Sensor Descriptions.

3. Test Set-up Parameters

(a) 15 minute simulation time.

(b) 100 runs (Monte Carlo Simulations).

Unless otherwise noted, the simulation parameters of Tables 2-4 are used throughout the simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION RESULTS

The plan position indicator display of the tracking results of Table 2 is included in Figure 2. This is a representative plot showing the location of the targets and the sensor platform. The same figure was produced for every test case that was run, but was not included as a figure for every case. Unless otherwise noted, the targets referred to in the rest of this section are the targets indicated in Figure 2.

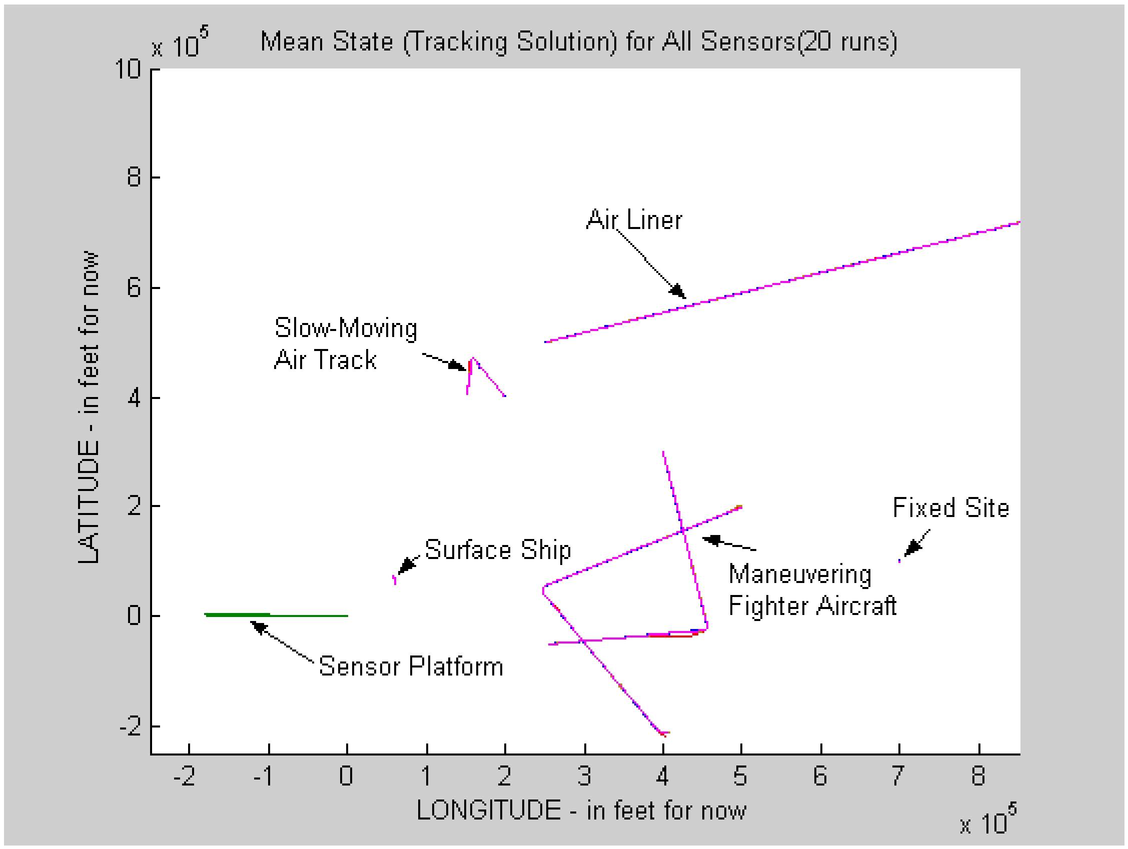

Figure 2. Plan Position Indicator (PPI) of Simulation Results.

A. TEST 1: TRACK ALGORITHM VALIDITY

Track Algorithm Validity Set (a) Results

To prove that the simulation was working for generation of truth data, measurement data and tracking data, the code was initially run with zero error input for the range and bearing errors. The sensor platform had zero error, the sensors each had zero error, and the tracker was allowed to produce a track and determine the Least Square Error compared to the truth data. With zero error in the sensor, several runs were conducted to demonstrate stability in the tracking codes. Under ideal conditions, with no error in the sensors, the track produced had zero error for straight line motion (Figure 3). Errors greater than zero were noted for maneuvering targets, but only in the turn. As expected, the errors quickly returned to zero upon completion of the turn.

Track Algorithm Validity Set (b) Results

For the next set of runs, the errors were reinstated and the noise terms in each of the trackers were adjusted to observe the differences in the tracking solution. For the PDAF, one clutter point was used in the algorithm. The number of clutter points was fixed at one to simplify the code and force the tracker to deal with a clutter point at every iteration. This would not be the case in an actual PDAF, where normally a test is performed at every measurement update to see if a clutter return originates from within the tracking gate. For the IMM, the straight line motion (SLM) model and turn model noise terms were adjusted to find a solution that performed well in the turn. The Kalman Filter (KF) and the Constant Gain Kalman Filter (CGKF) were not evaluated during this portion of the test.

Test 1: Set (a)	**Tracking Algorithm Validity**			
Description:	Determine if the tracking algorithms are valid with zero tracking error			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

Note: The test conditions are highlighted in each figure test table.

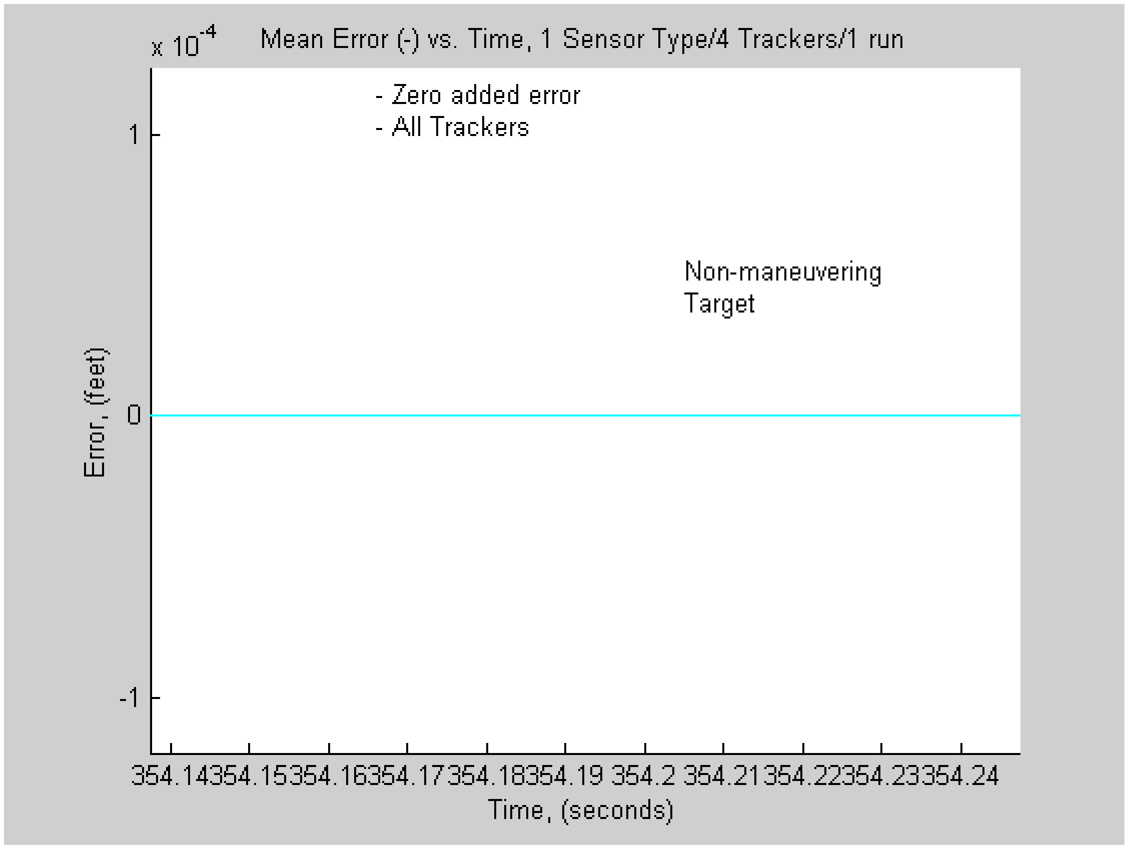

Figure 3. Tracking Solutions with no Measurement Errors.

The airborne surveillance radar was used to check the validity of the PDAF. The PDAF was isolated as a tracker, the noise term was adjusted to intentionally force the tracker to miss the turn, and the results were plotted to demonstrate the subtle weakness of the PDAF. The PDAF worked well for tracks in straight-line motion with $q^2 = 1.0$, but lost the track in the turn (Figures 4 and 5). Instead of updating the track state with the measurement, the track state was updated with a more heavily weighted clutter point. At $q^2 = 100$ (Figure 6), the PDAF was able to maintain a reasonable track through the turn for this set of runs. However, for higher accelerations or turn rates, this PDAF setup could not be expected to maintain the track. In fact, the model runs a rather low turn acceleration (1.5g) that does not represent a worst-case scenario. For this simple implementation of the PDAF, it is always possible for the track to get pulled off by the clutter point, mainly because of the previous assumption: The clutter point always falls within the tracking gate (recall that the tracking gate is centered around the predicted position of the target).

The mean measurement noise appears to be dropping off rapidly in each of the figures. This is due to the change in geometry from the moving target and sensor platform. At $q^2 = 250$ (Figure 7), the PDAF tracked fine through the turn, but the mean measurement error was about 400 feet higher for the same target. These adjustments for the noise term are the same as the noise adjustments in the Kalman filter. For the next set of runs, the noise term was adjusted to allow the PDAF to track through turns in search of an optimum solution. At $q^2 = 250$, the average track error was up around 700 feet for SLM, but down around 300 feet with $q^2 = 1$. To ensure a constant turn performance throughout the rest of the tests, the PDAF was run with a noise term of $q^2 = 1000$ (unless otherwise noted).

For the IMM, the two models were varied by simply altering the noise term in each model. For the straight line motion model, the noise term was very small to take advantage of the SLM performance of the Kalman filter. The Markov probabilities were 0.1 for the turn probability, and 0.33 for the stop turn probability. Changing these probabilities do affect the performance of the filter, but this aspect of the IMM was not rigorously examined in this thesis. The IMM was run for several different cases with varying values of the noise term. For the turn model, the noise term was gradually increased until the filter performed consistently well against a maneuvering target. Only two of the cases are displayed in Figures 8 and 9. Reasonable performance was obtained in the turn with the SLM and turn $q^2 = 1$ and 10000 respectively.

The noise terms in the Constant Gain Kalman Filter (CGKF) and the Kalman Filter (KF) were not evaluated for the purpose of this Thesis due to proven performance in other exercises [Ref. 9].

Test 1: Set (b)	**Tracking Algorithm Validity**			
Description:	Determine an applicable noise term (q^2) for follow-on tests Demonstrate susceptibility to losing the track in a turn of the PDAF with a low noise term			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- $q^2 = 1.0$ - 1 clutter pt	- SLM $q^2 = 1.0$ - Turn $q^2 = 10000.0$	- $q^2 = 1000.0$	- $q^2 = 1000.0$
Targets	Maneuvering Aircraft (2)	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

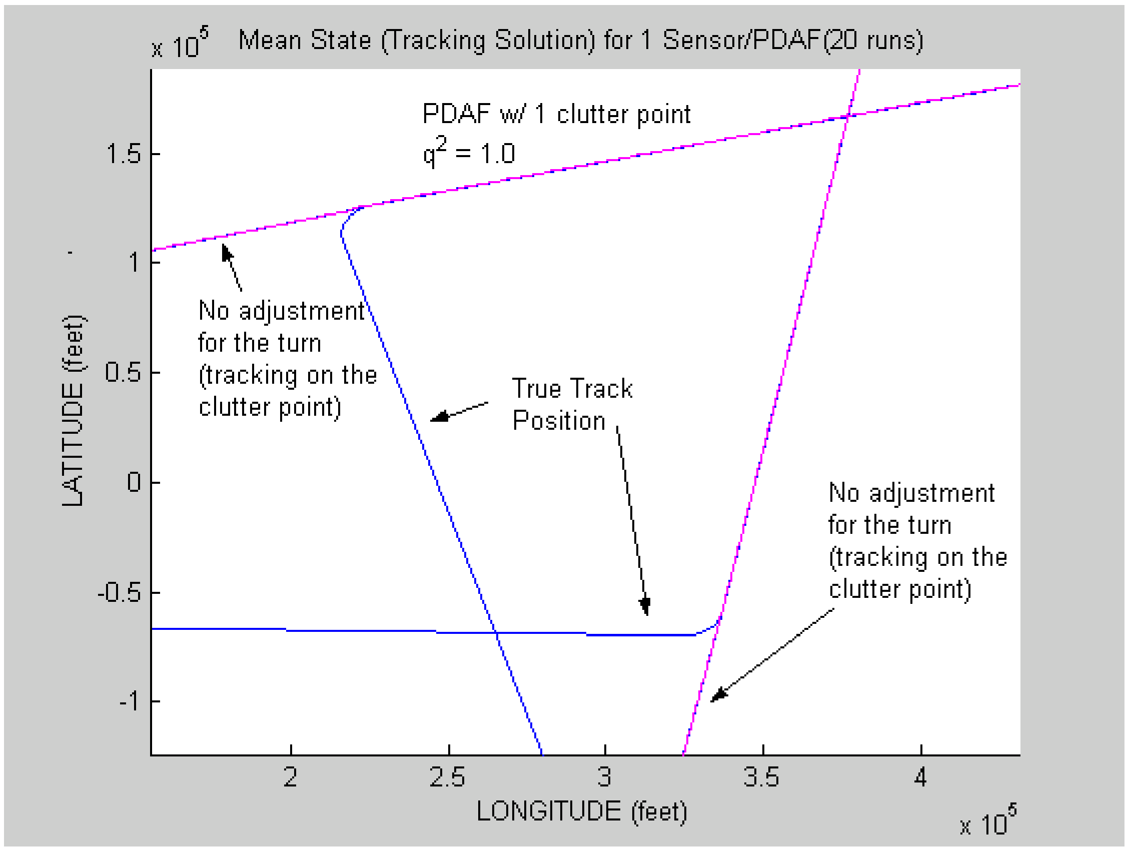

Figure 4. Tracking Results for PDAF with Low Noise Term.

Test 1: Set (b)	Tracking Algorithm Validity			
Description:	Determine an applicable noise term (q^2) for follow-on tests Demonstrate susceptibility to losing the track in a turn of the PDAF with a low noise term			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

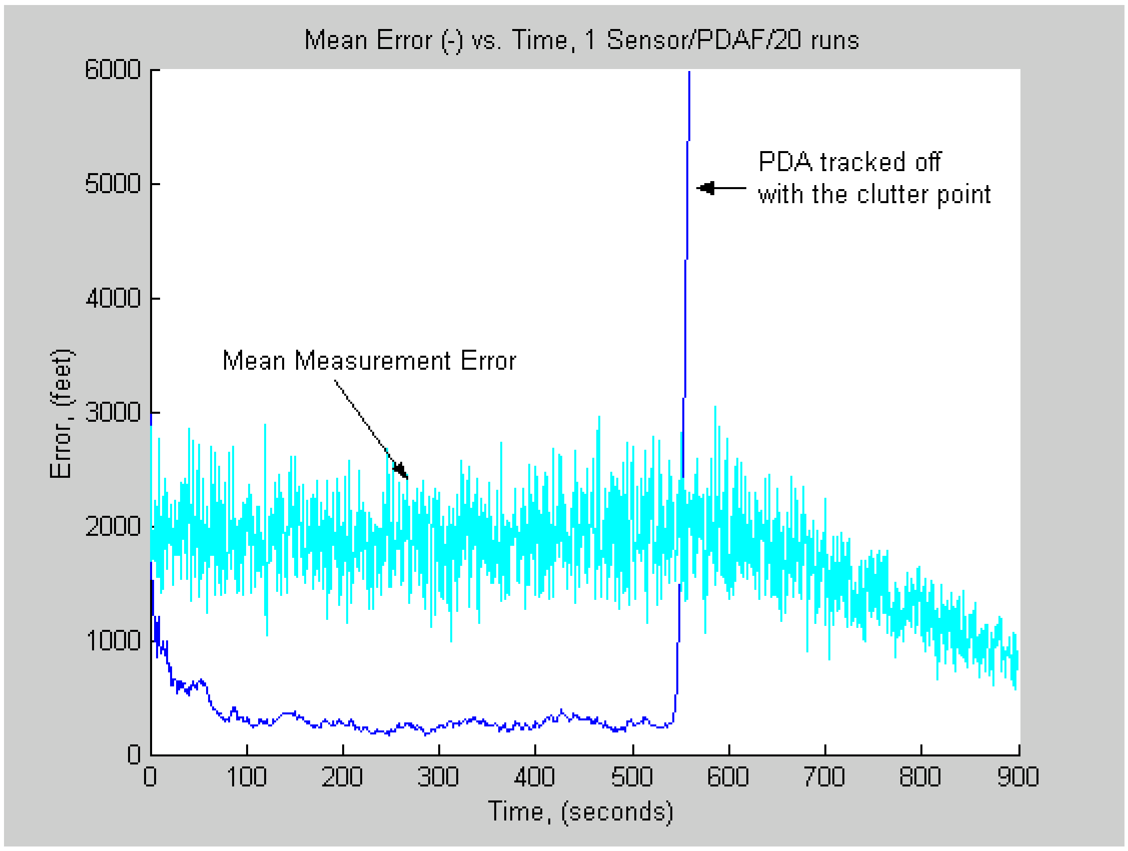

Figure 5. PDAF with Low Noise Term.

Test 1: Set (b)	**Tracking Algorithm Validity**			
Description:	Determine an applicable noise term (q^2) for follow-on tests Demonstrate how increasing the noise term improves the turn performance of the PDAF			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 100.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

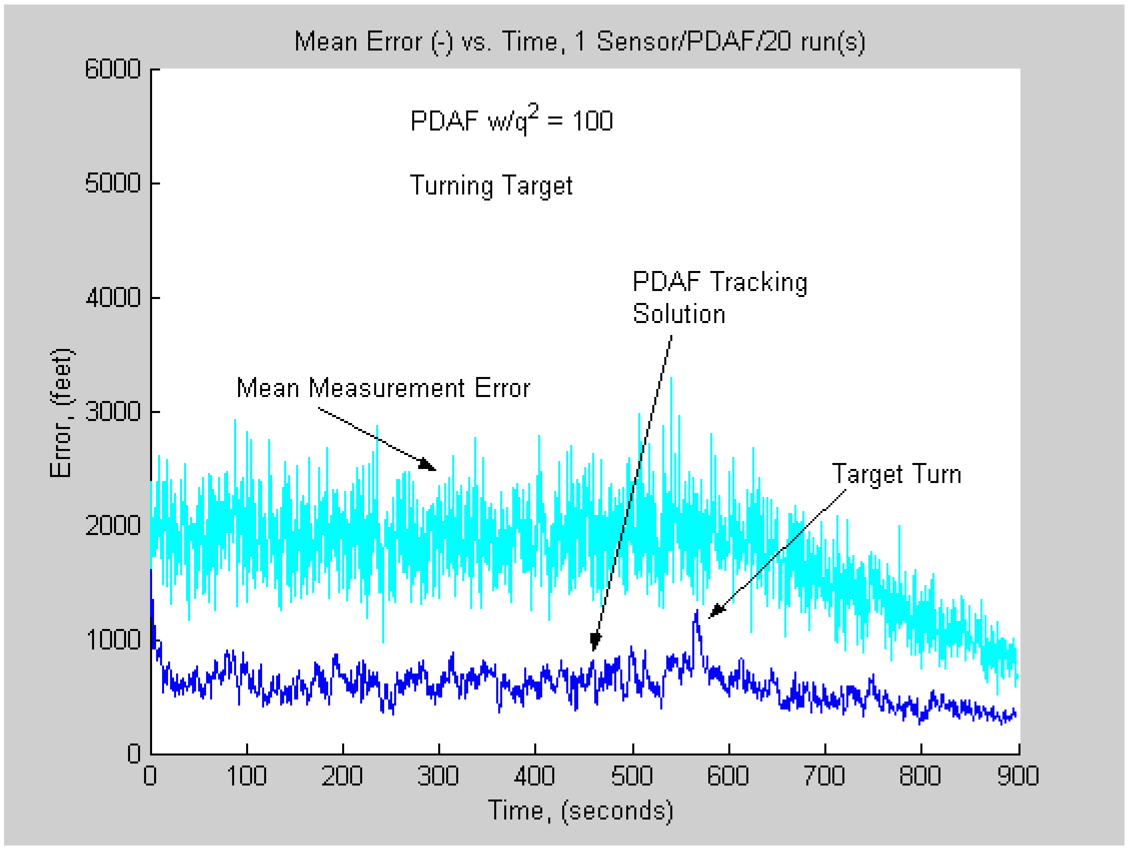

Figure 6. PDAF with Increased Noise Term.

Test 1: Set (b)	**Tracking Algorithm Validity**			
Description:	Determine an applicable noise term (q^2) for follow-on tests Demonstrate how increasing the noise term improves the turn performance of the PDAF			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 250.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

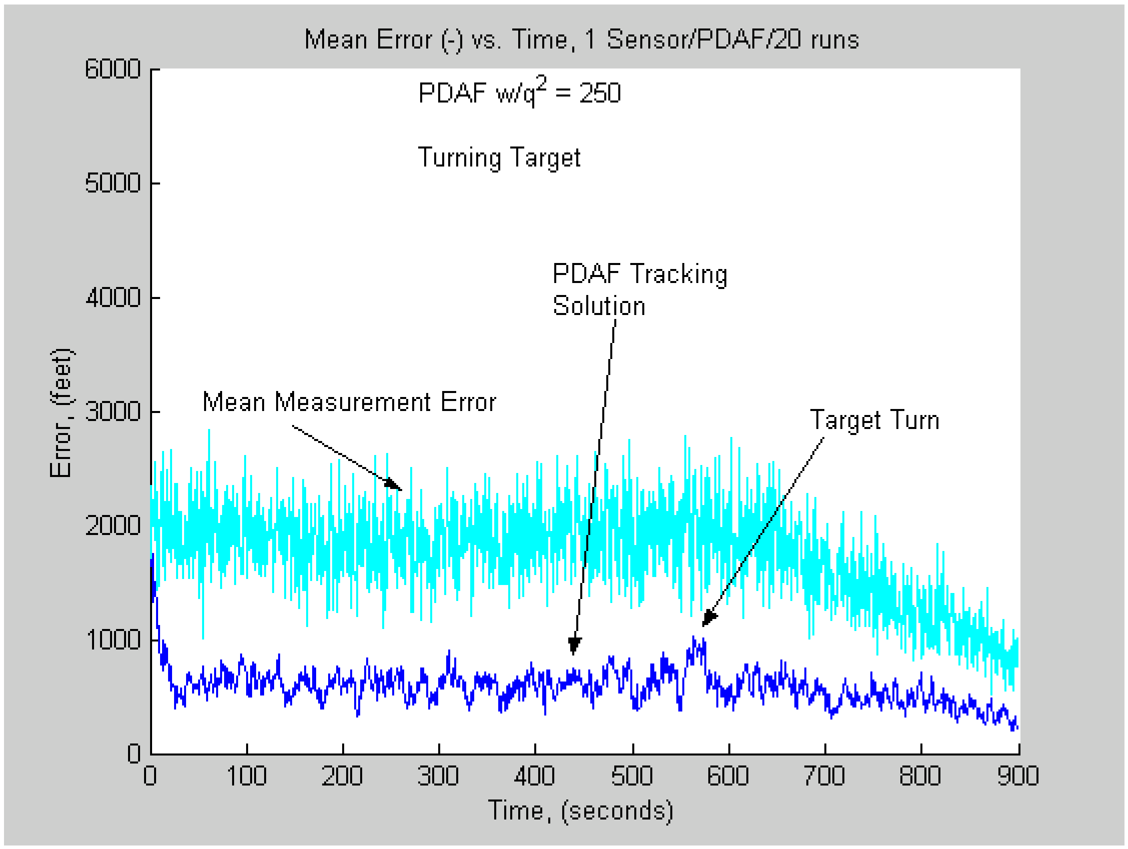

Figure 7. PDAF with Increased Noise Term.

Test 1: Set (b)	**Tracking Algorithm Validity**			
Description:	Determine an applicable noise term (q^2) for follow-on tests Demonstrate how changes in the models affects the IMM performance.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 250.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

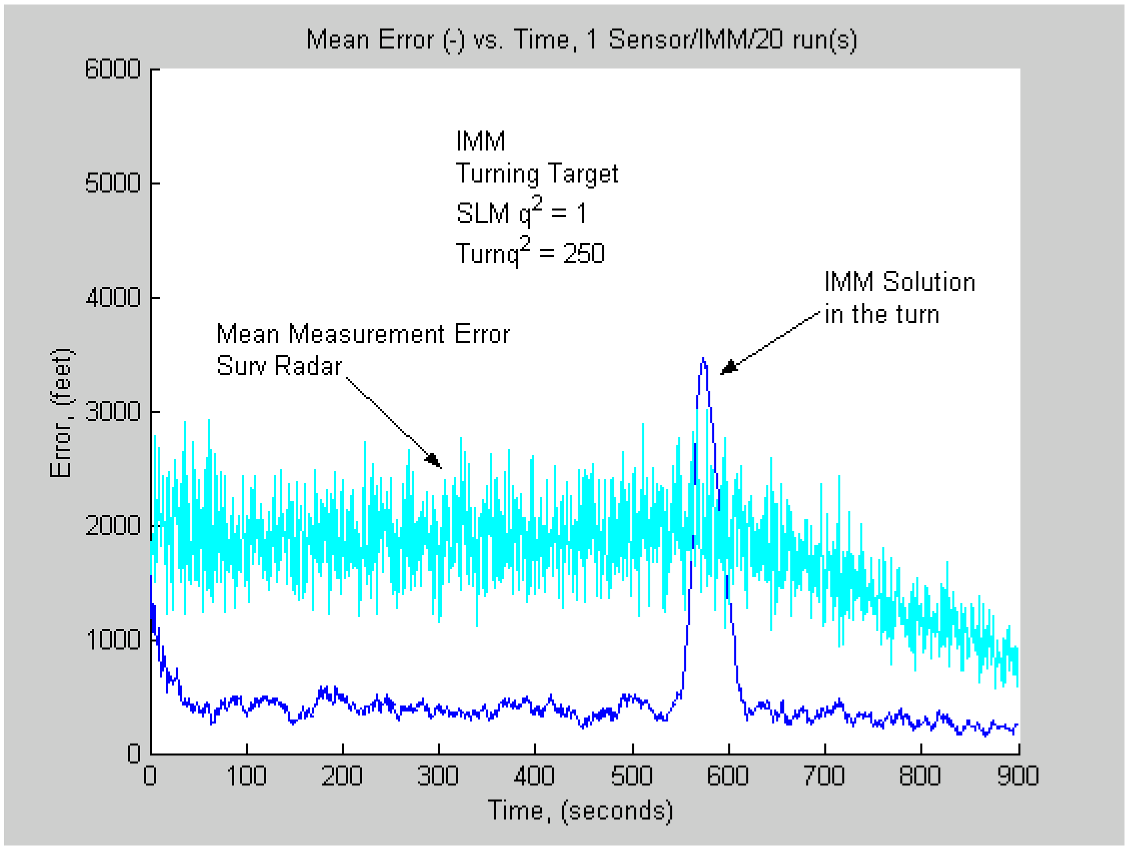

Figure 8. IMM Tracking Performance with Varying Models.

Test 1: Set (b)	**Tracking Algorithm Validity**			
Description:	Determine an applicable noise term (q^2) for follow-on tests Demonstrate how changes in the models affects the IMM performance.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

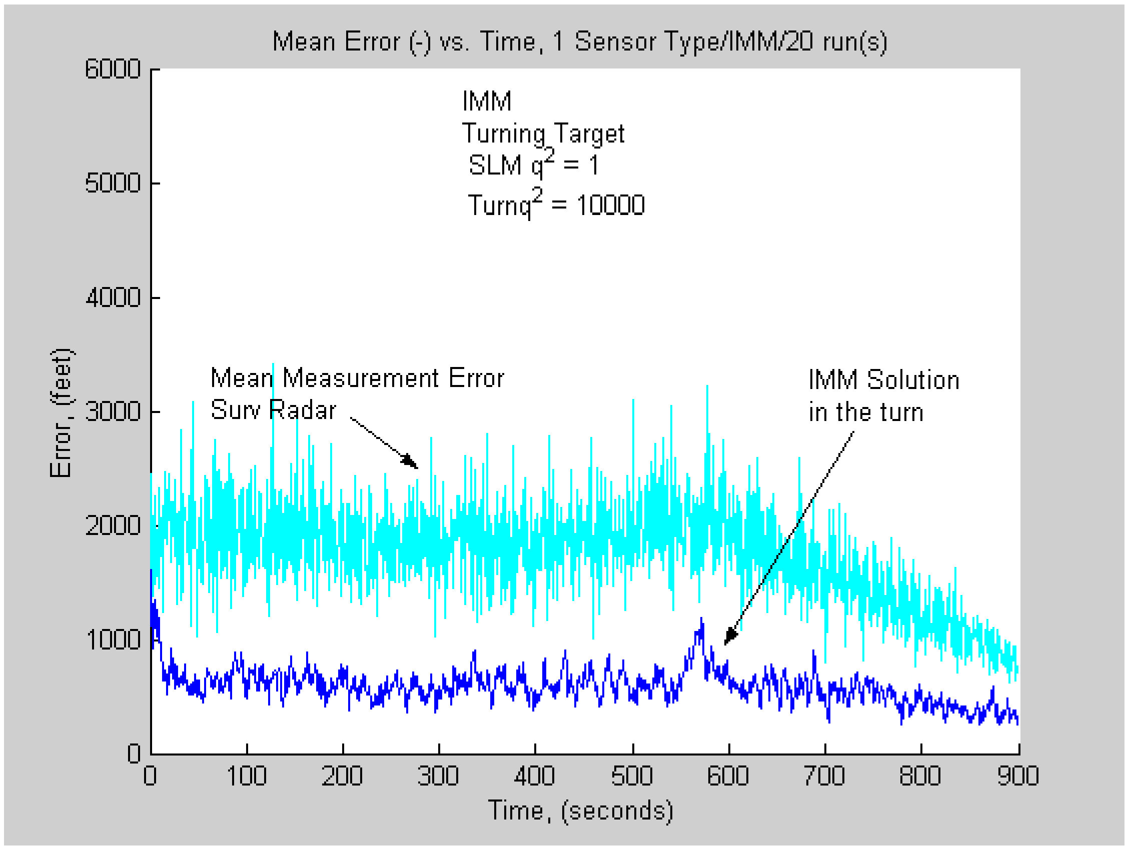

Figure 9. IMM Tracking Performance with Varying Models.

Track Algorithm Validity Set (c) Results

The airborne surveillance radar was the first sensor to be evaluated with all 4 trackers (Figures 10 through 13). For the maneuvering target case of Figure 10, the trackers all performed reasonably well. However, the CGKF took far too long to establish the track The large initial error (>6000ft in this case) is a characteristic of the CGKF. The only tracker with an advantage in this case was the IMM. It was slightly worse in SLM and slightly better in the turn. It would be easy to force the KF as the optimum solution in SLM by reducing the noise term to 1, but turn performance would suffer greatly. Also, the IMM could be forced to perform better in the turn by increasing the turn noise term. However, the SLM performance would be further degraded.

In the case of the non-maneuvering target (Figure 11), the PDAF and the Kalman filter were about the same. The IMM was slightly worse mainly because of the expectation of a turn. The CGKF performed adequately, but errors in the CGKF increased much faster than the other trackers as range to the target and mean measurement error increased. The mean measurement errors increased over time due to increasing range from the sensor platform.

In the case of the slow-moving ship (12 knots), the PDAF immediately picked up on the clutter point and tracked off (Figure 12). Some parameters may need to be changed in the PDAF for tracking slow moving targets. The PDAF may not be well suited for tracking slow moving and stationary targets in a clutter region without modifications to make the code more adaptable.

For the fixed site (Figure 13), the trackers all seemed to perform about the same for the surveillance radar, even though the mean measurement error was considerably higher. The IMM error was slightly higher throughout, but that may still be due to the turn expectation and the high q^2 of the turn model.

Test 1: Set (c)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

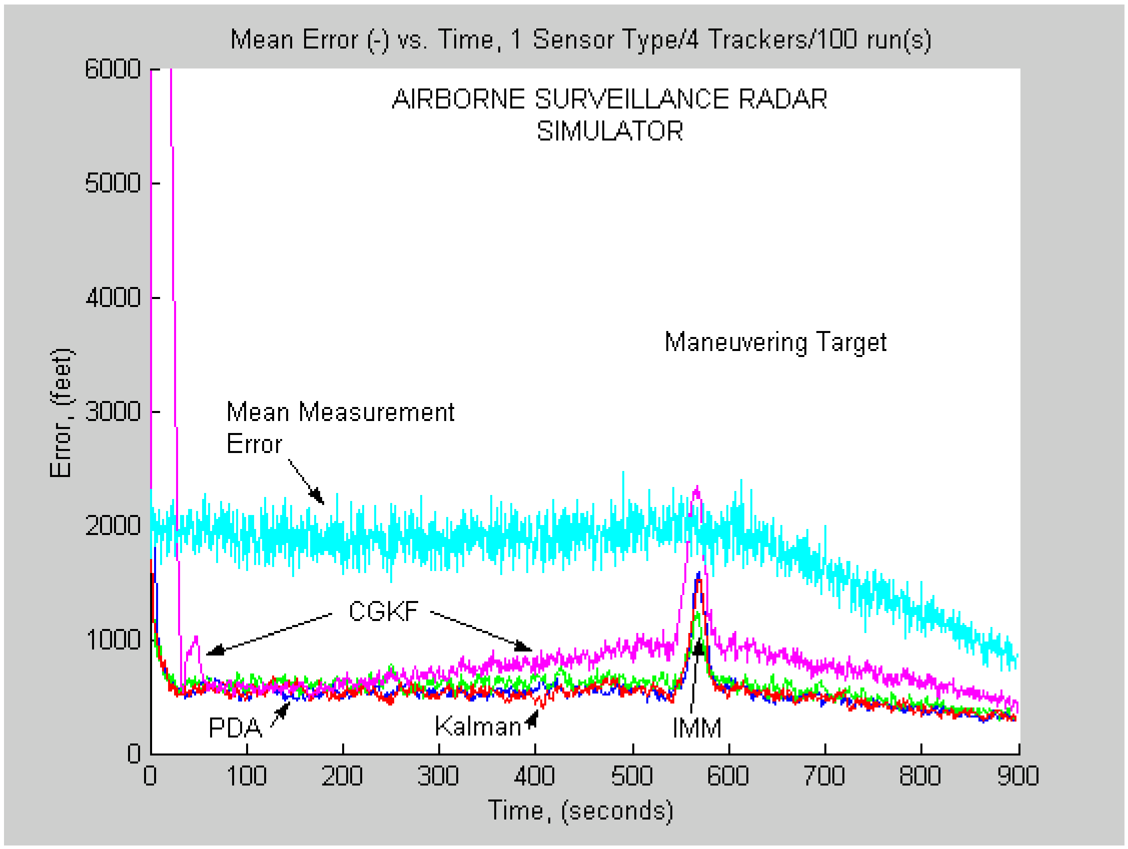

Figure 10. Tracker Comparison with One Sensor Type for the Maneuvering Target.

Test 1: Set (c)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

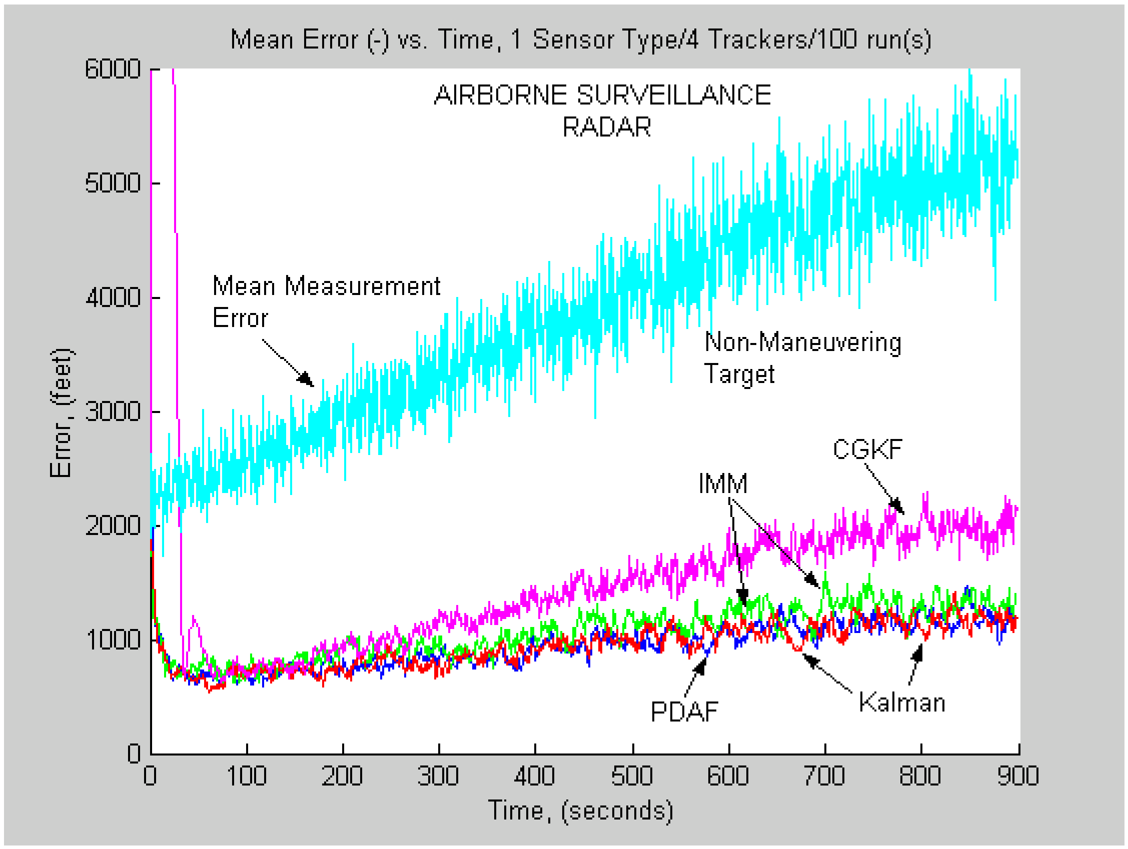

Figure 11. Tracker Comparison with One Sensor Type for the Non-maneuvering Target.

.

Test 1: Set (c)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

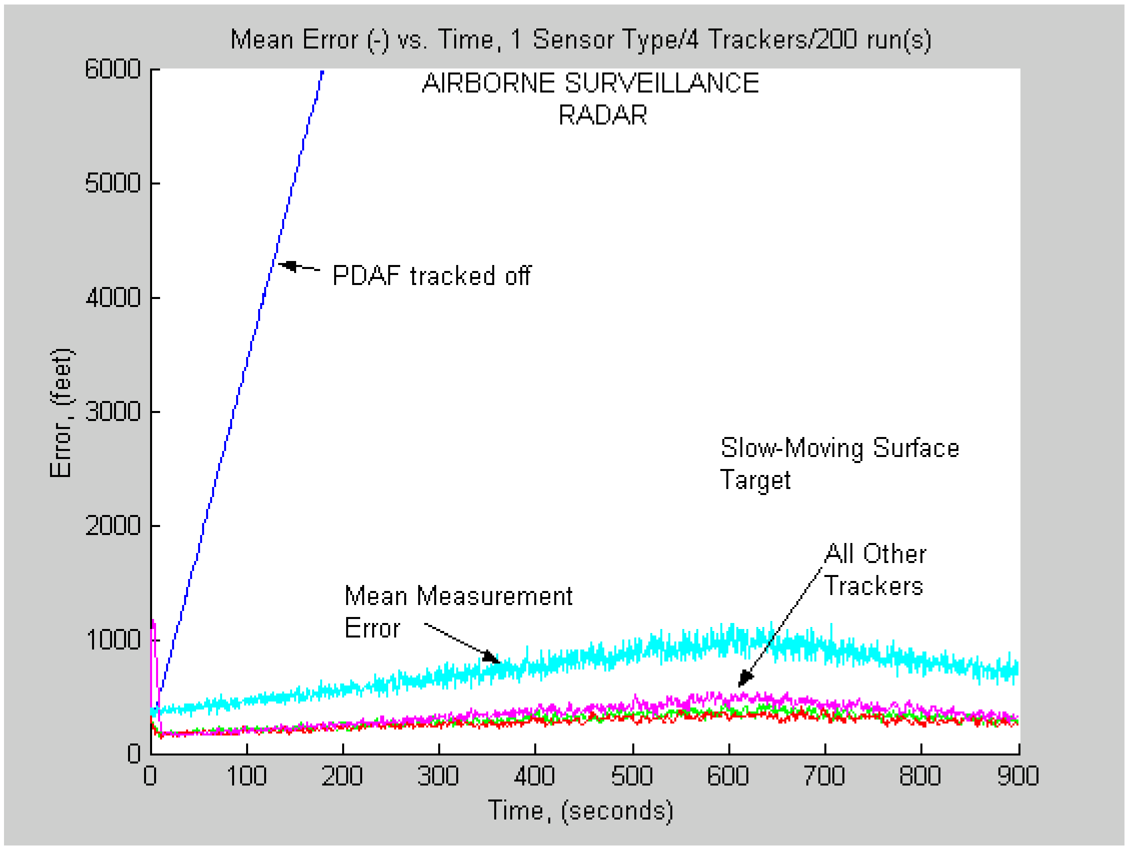

Figure 12. Tracker Comparison with One Sensor Type for the Slow-moving Ship.

Test 1: Set (c)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

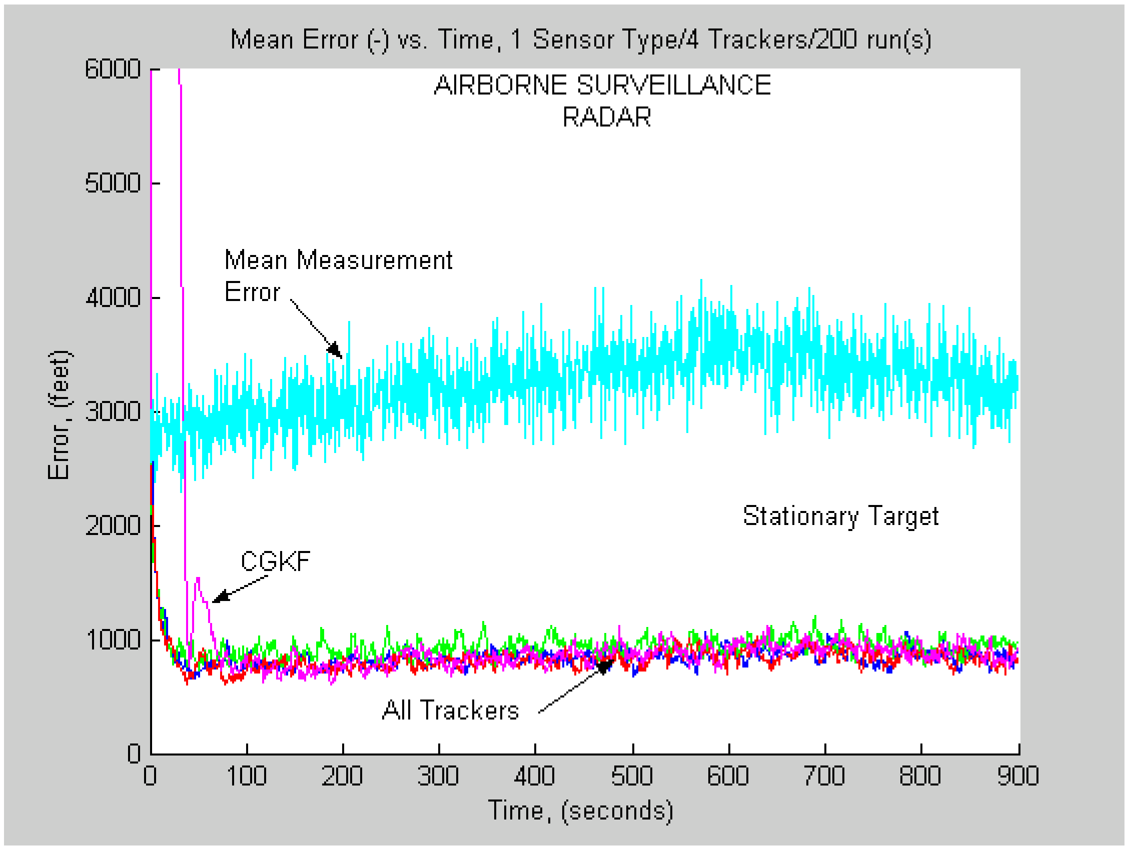

Figure 13. Tracker Comparison With One Sensor Type for the Fixed Site Target.

Track Algorithm Validity Set (d) Results

For Set (d), the surface fire control radar was examined to look for anomalies in the trackers when paired with a highly accurate sensor (Figure 14). The ranges and accuracy's of the sensor may be unrealistic, but there was no apparent advantage to any of the trackers with a sensor this accurate. In other words, the simple trackers worked just as well as the more complex trackers. This phenomenon gives the appearance that the state prediction and estimation was a waste of time and computing power. However, estimation is still necessary for predicting the state of the target, for tracking a maneuvering target, and for guiding a missile or projectile to an impact point. For the purposes of this test, the low errors were used to look for anomalies in the trackers.

Test 1: Set (d)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Fire Control Radar Simulator	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

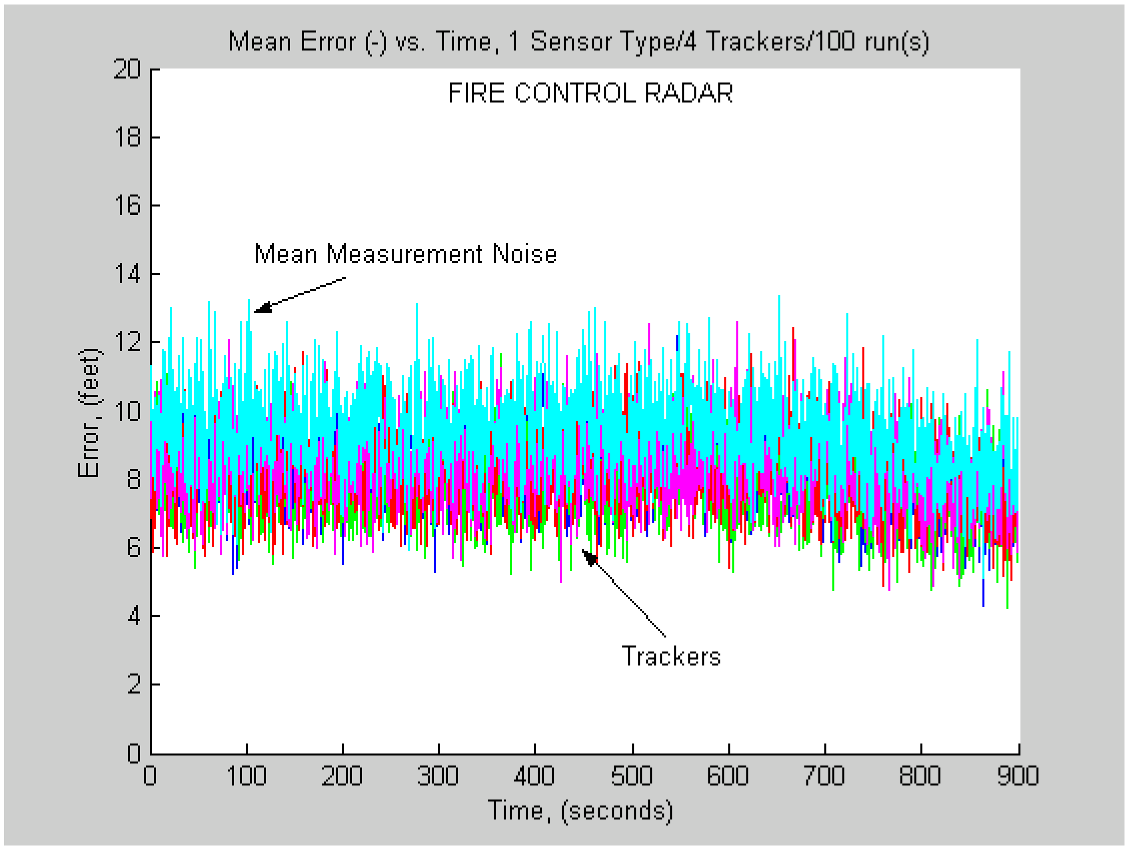

Figure 14. Tracker Comparison with One Sensor Type for the Maneuvering Aircraft.

Track Algorithm Validity Set (e) Results

The last test case in the first set of runs was the Infrared Search and Track (IRST) simulator. In formulating this very simple model of the an IRST system, the bearing to the target was assumed to be very accurate and the range was assumed to be very ambiguous. In a true IRST system, the range is not measured by the system, unless by some other means. To keep this analysis simple, it is assumed that the IRST has already established a track, and the measurement is based on the position report from the tracker.

For the maneuvering target (Figure 15), the higher mean measurement noise of the IRST system made little difference in the tracking performance. The CGKF error steadily increased as range to the target increased. An error increase in the other trackers was imperceptible. The IMM handled the turn fairly well, yet the Kalman and the PDAF both did worse in the turn.

Test 1: Set (e)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

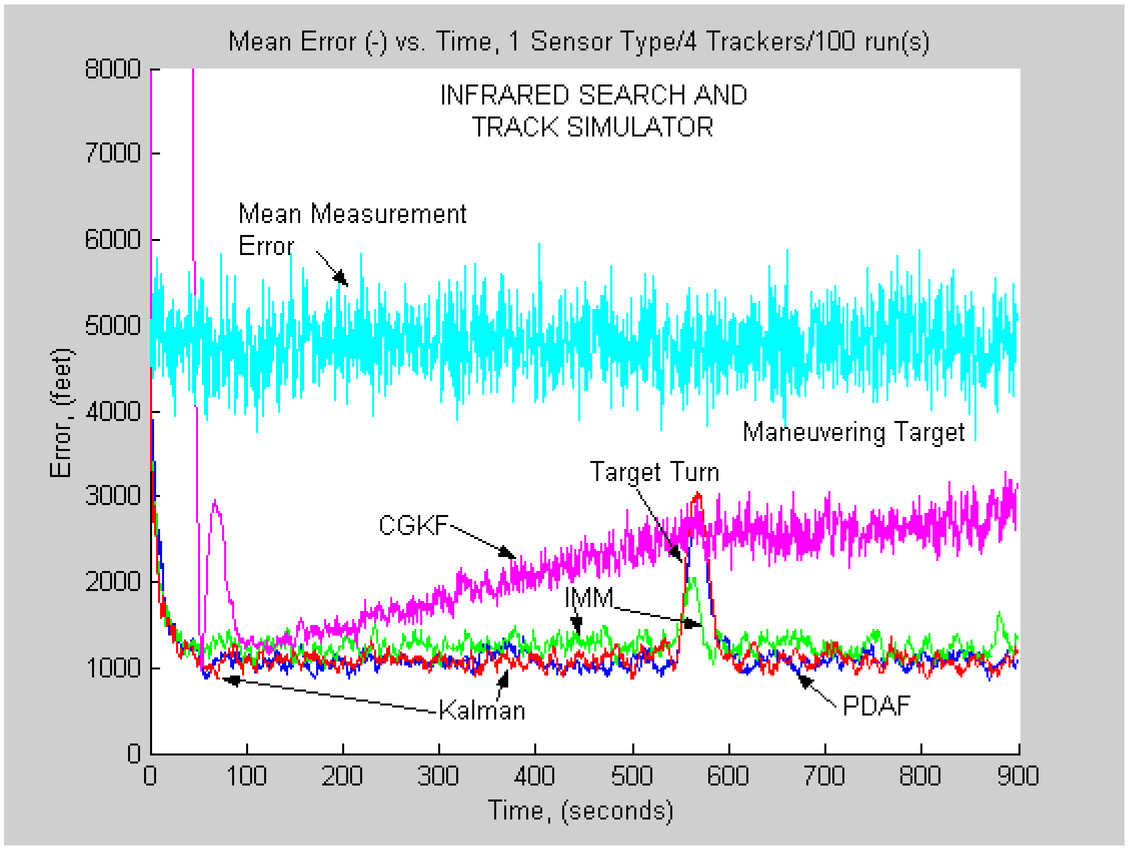

Figure 15. Tracker Comparison with One Sensor Type for the Maneuvering Aircraft.

For the non-maneuvering target (Figure 16), the results were similar to the maneuvering target case. The CGKF error gradually increased, while the other trackers remained steady around 1000 feet of error. For the slow moving ship of Figure 17, the CGKF error seemed to vary with distance from the sensor platform, even though the measurement error remained about the same over the simulation period. The PDAF, the IMM, and the KF all hovered around 1000 feet of error.

In this case of the fixed site (Figure 18), the PDAF lost the track, yet the other trackers did consistently well. The clutter point may have caused sufficient variation in the state estimate to induce a velocity and cause the solution to track off. This problem did not occur every time with the PDAF and may indicate a higher degree of uncertainty in a PDAF solution. To handle a slow-moving or non-moving target in a clutter region, some modification within the PDAF would be required to help classify the target and allow better estimates. For example, a velocity threshold could be used to help decide the class or type of target, so that unrealistic velocity estimates could be eliminated from the solution.

Test 1: Set (e)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

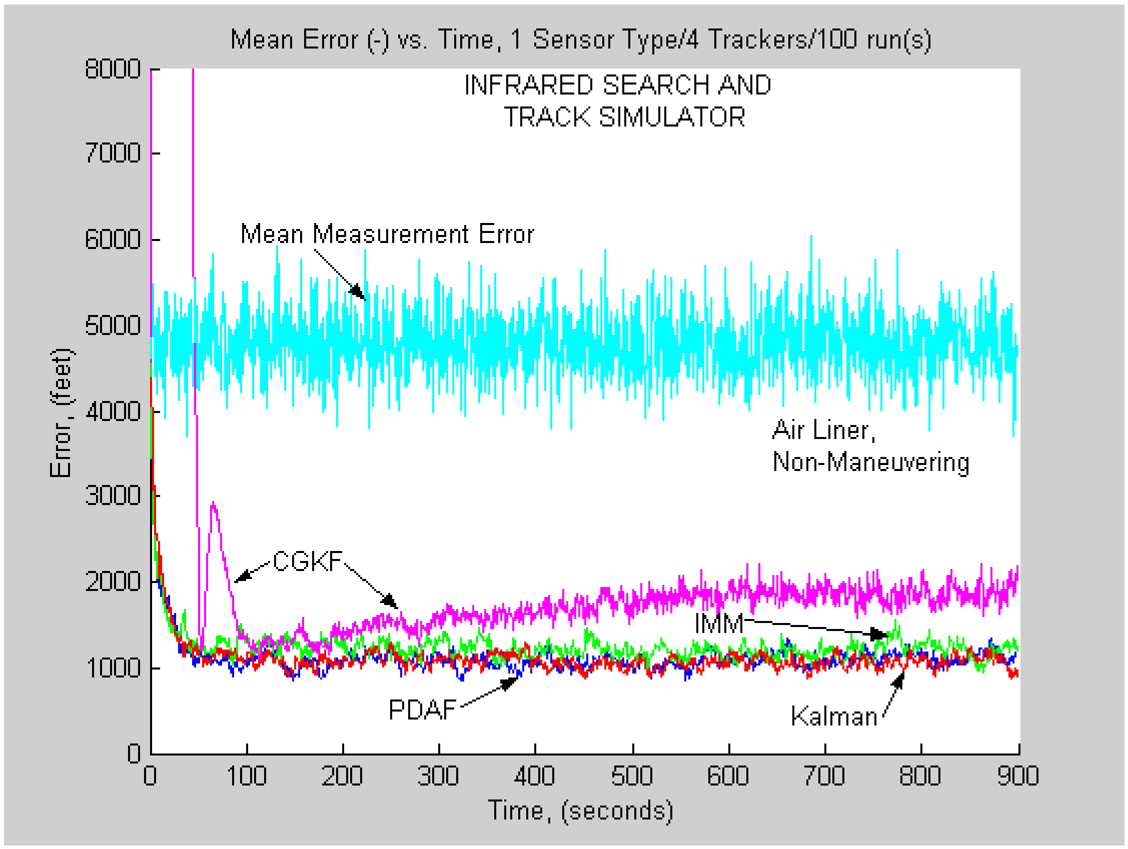

Figure 16. Tracker Comparison With One Sensor Type For The Non-Maneuvering Aircraft.

Test 1: Set (e)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- $q^2 = 1.0$ - 1 clutter pt	- SLM $q^2 = 1.0$ - Turn $q^2 = 10000.0$	- $q^2 = 1000.0$	- $q^2 = 1000.0$
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

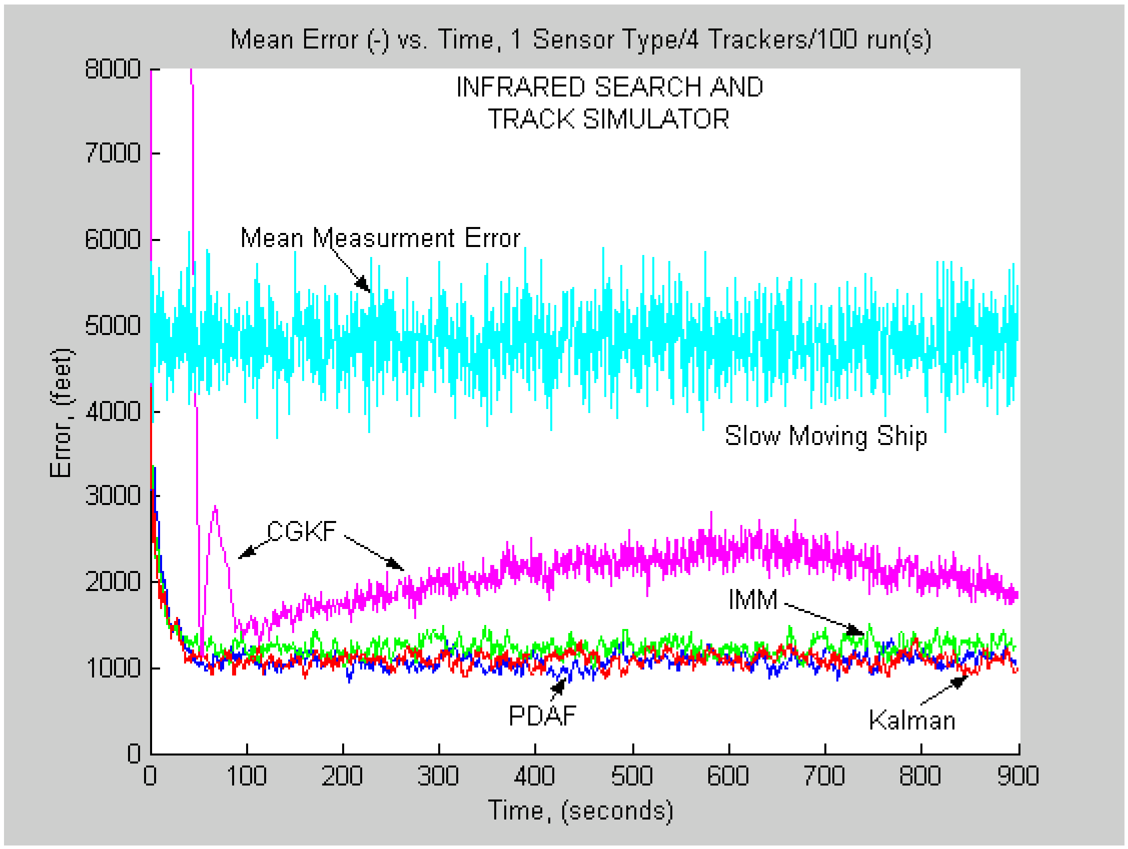

Figure 17. Tracker Comparison With One Sensor Type For The Slow-Moving Ship.

Test 1: Set (e)	**Tracking Algorithm Validity**			
Description:	Observe the performance of the trackers using the same sensor type with each of the trackers			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

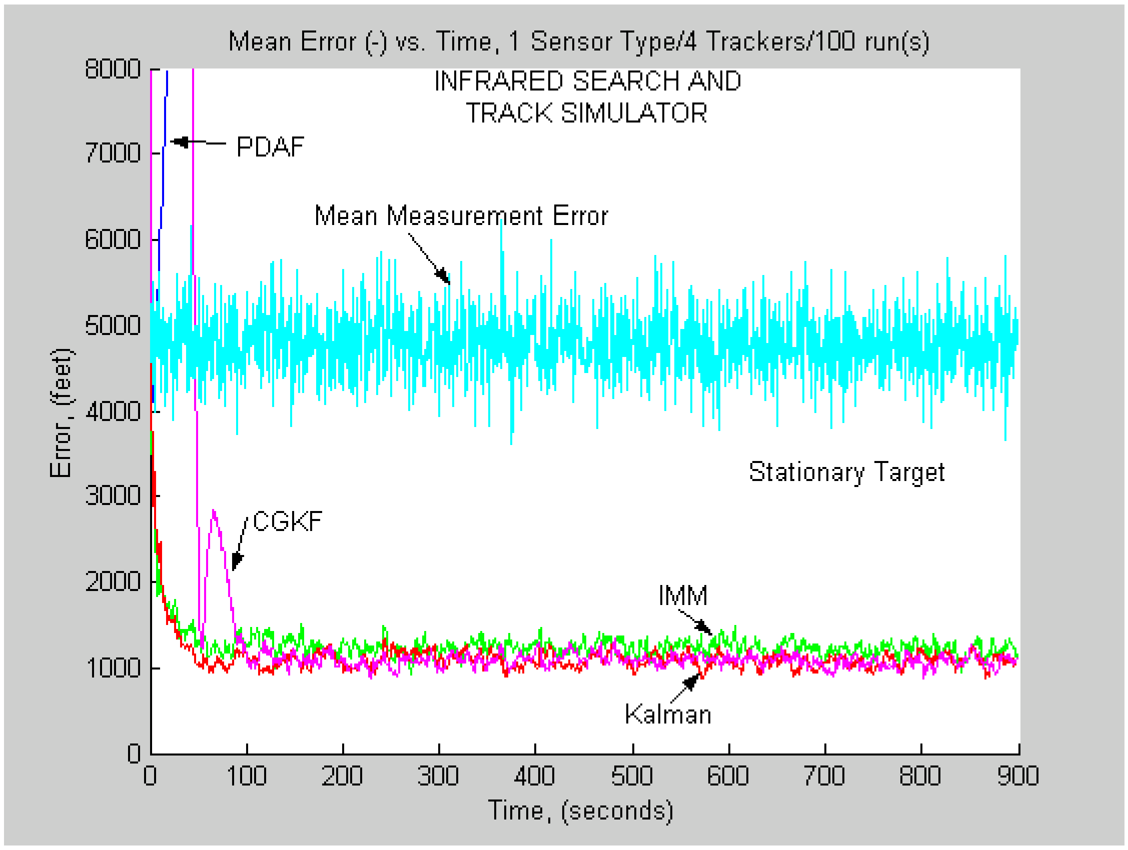

Figure 18. Tracker Comparison with One Sensor Type for the Fixed Site Target.

B. TEST 2: MULTIPLE SENSOR MULTIPLE TRACK

For Test 2, the multiple sensor/multiple track case was examined to demonstrate the different capabilities of sensors and trackers. Representative plots for the maneuvering target were included. The sensors and trackers were paired up as shown in Table 5.

Airborne Surveillance Radar	Probabilistic Data Association Filter (PDAF)
Identification Friend or Foe Detection System (IFF)	Interacting Multiple Models (IMM) 2-model filter
Electronic Surveillance Measures System (ESM)	Kalman Filter (KF)
Infrared Search and Track System (IRST)	Constant Gain Kalman Filter (CGKF)

Table 5. Sensor/Tracker Pairs.

Figure 19 is a display of the mean measurement errors in each of the sensors, while Figure 20 is the tracking solution for each sensor for the same run. Again, this test was simply a demonstration of the diversity in sensor capabilities. By observation, the trackers are sensitive to the amount of error in the sensors. For example, the IMM solution tracked very well due to the low errors in the IFF sensor.

Some fine tuning in the filters could help improve tracking performance. However, a trade-off exists when adjusting the noise term, q^2. If the noise term is increased to reduce errors in the turn, the SLM performance will suffer as shown in Test 1. This concept will be demonstrated again in Test 3, Set (c) by increasing the noise term in each tracker.

Test 2: Set (a)	Multiple Target/Multiple Sensor			
Description:	Demonstrate the differences between sensors, their associated mean measurement errors, and the resulting tracking solution.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

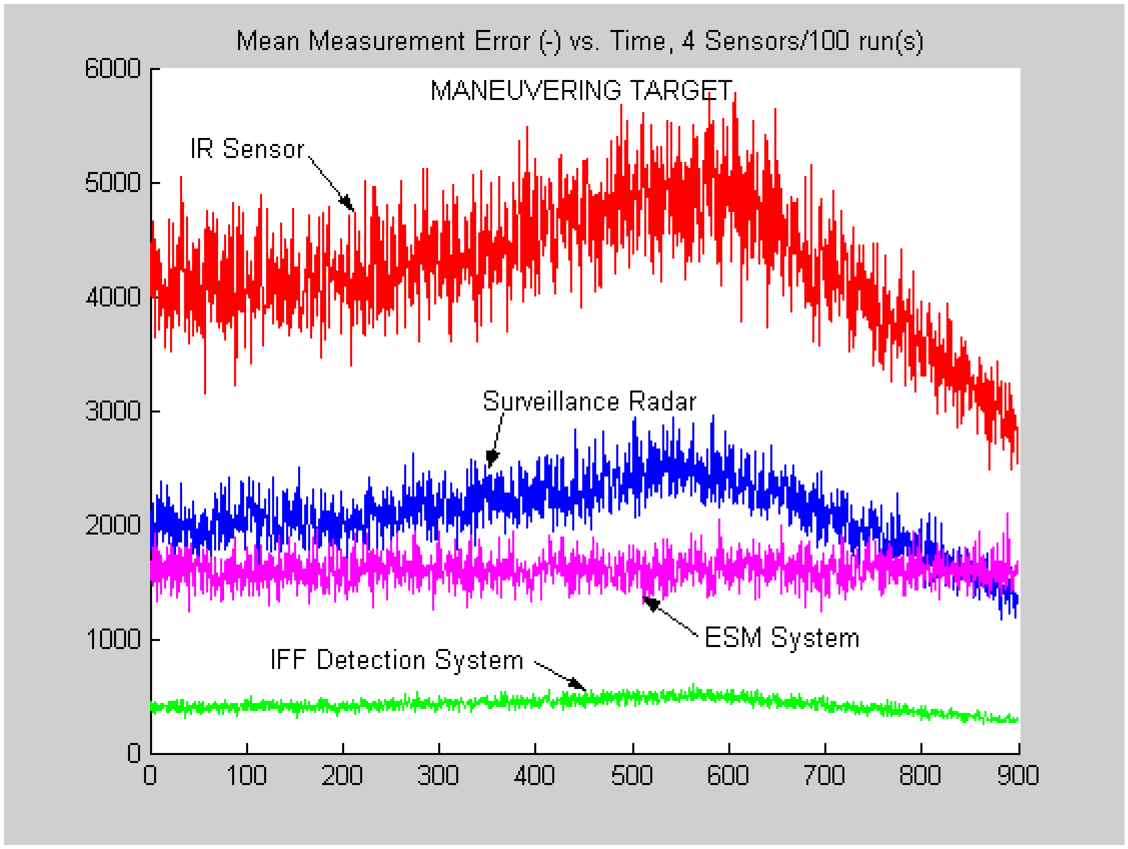

Figure 19. Mean Measurement Error for Varying Sensor Types.

Test 2: Set (a)	**Multiple Target/Multiple Sensor**			
Description:	Demonstrate the differences between sensors, their associated mean measurement errors, and the resulting tracking solution.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 1000.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

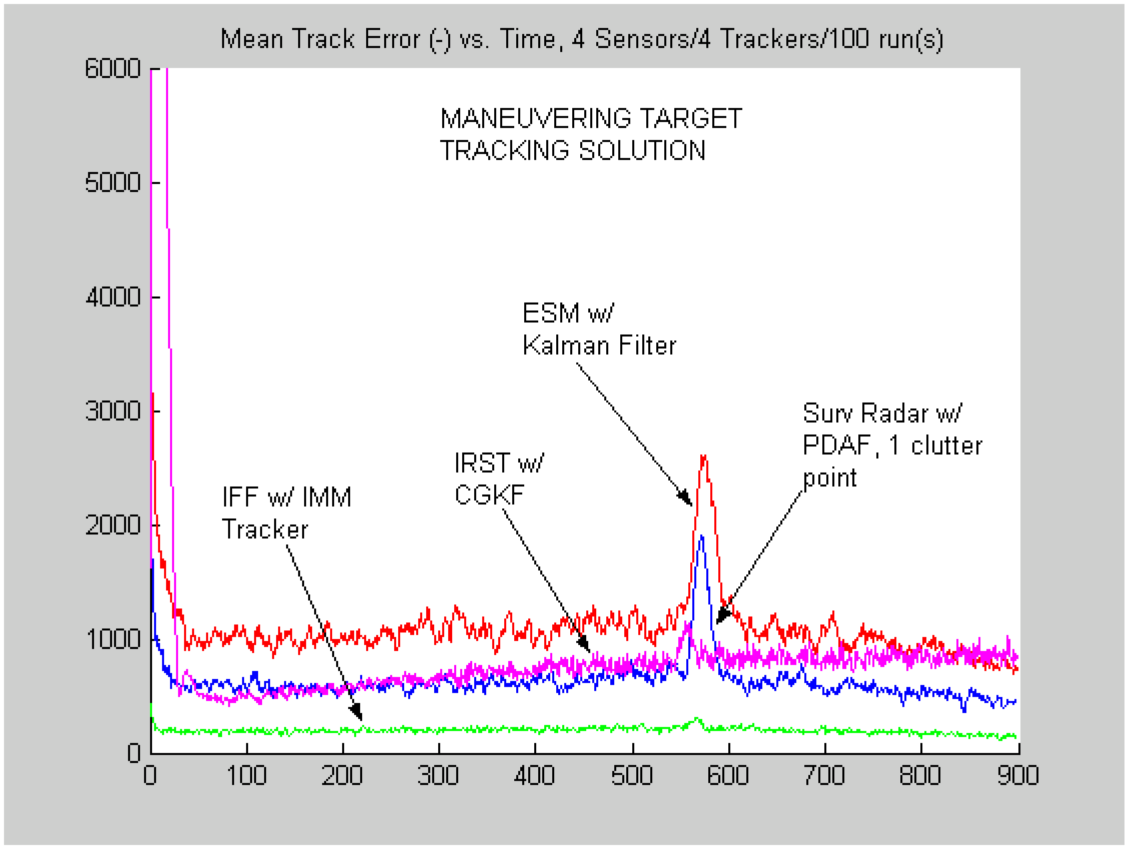

Figure 20. Tracking Solutions for the Multiple Sensor Case.

C. TEST 3: OBSERVATIONS ON ALGORITHM BEHAVOIR IN A SIMPLE COMBINED SOLUTION

The purpose to Test 3 was to provide a demonstration of a multiple sensor system output, and to gain insight into the behavior of a combined tracker estimate. The combined solution was based on weights that were empirically derived from the results of Test 1, and the multiple sensor simulation results of Test 2. No additional data sources were used in determining the solution. The combined state result is simply the weighted average of each of the tracker solutions. This increases the amount of data stored by the program. For the *fusim* simulation, the storage matrices and the function calls were a first step in implementing a track-to-track data association algorithm. Again, for this section, no additional filtering and no data association occurs in calculation of the weighted average of the track states. A true sensor fusion algorithm will be much more complicated [Ref. 1].

Combined Solution Set (a) Results

The simple combined solution has initial conditions as described in Test 1 for the single sensor case. The output of the simulation is included in Figure 21. One sensor type is used with the four trackers to compare the trackers and to observe the results of combining the state outputs of each tracker in a rudimentary fashion. To further demonstrate the effects of diversity in the sensors and tracking algorithms, the Kalman filter was given a low noise term ($q^2 = 2$) to allow an optimum solution in SLM.

In the turn, only one of the 4 sensor/tracker pairs reported high errors (Figures 22 and 23). This effect was by design due to the reduced noise factor, q^2 in the Kalman

Filter. The overall error was significantly reduced by combining the results of the different systems. The non-maneuvering target (Figure 24) had the best results from the Kalman Filter, since the filter was optimized for straight line tracking.

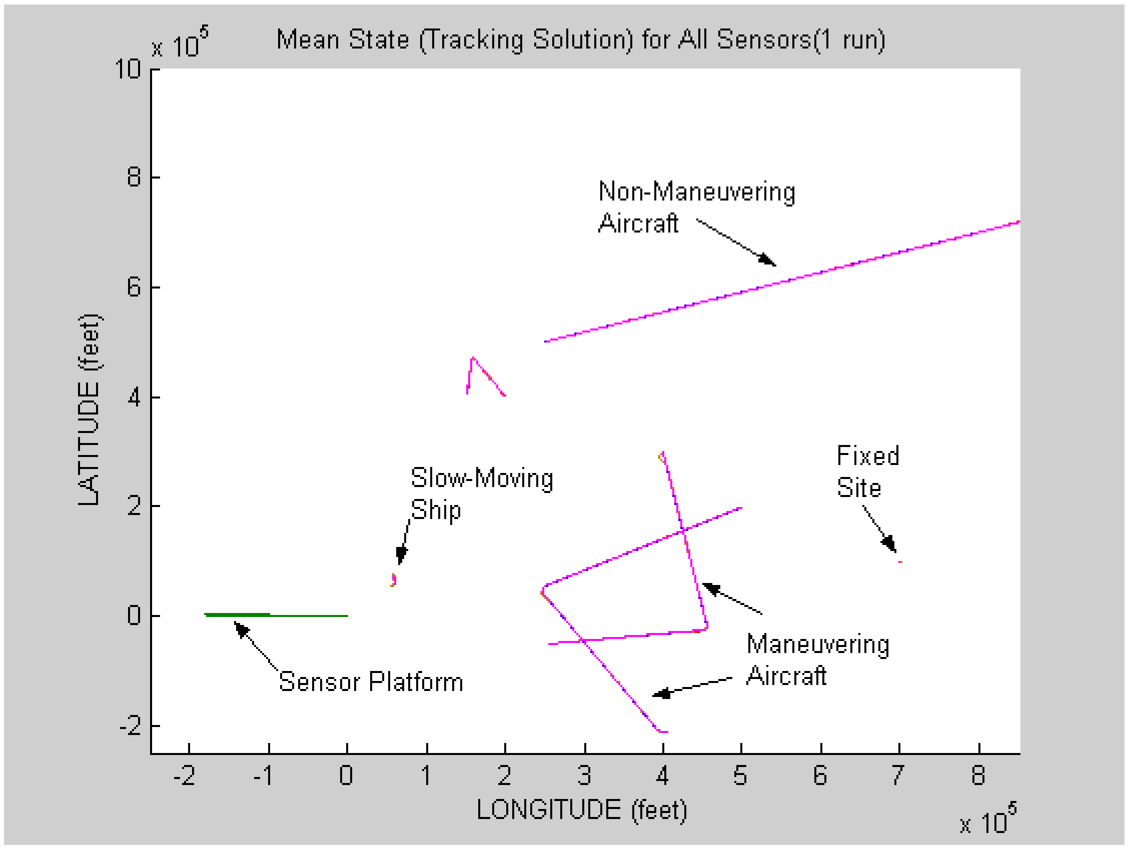

Figure 21. Sample Plot of the Combined Solution.

For the stationary target (Figure 25), the mean measurement error was greater due to the distance from the sensor platform. Again, the Kalman filter performed the best even though the sensor errors were high. The combined solution represents a compromise.

Test 3: Set (a)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using a single sensor type.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

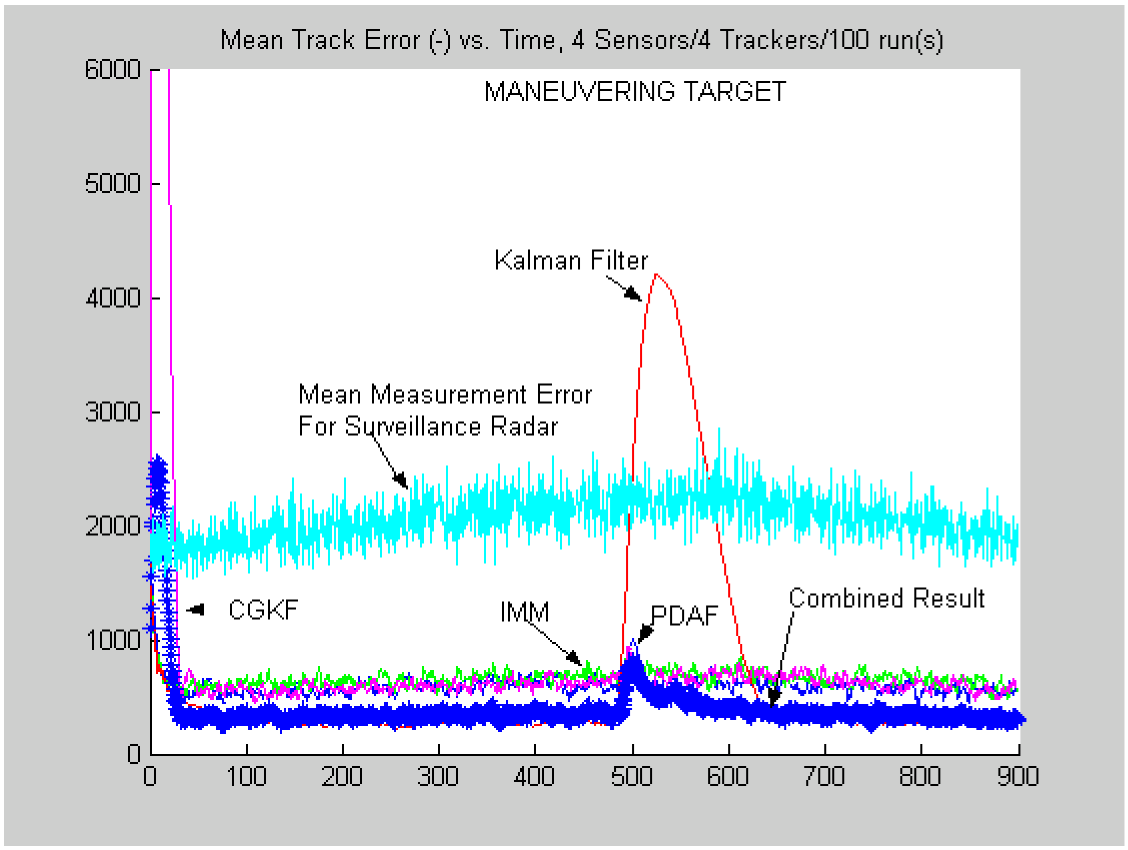

Figure 22. Simple Combined Solution for Maneuvering Target.

Test 3: Set (a)	Simple Combined Solution for Multiple Target/Multiple Sensor			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using a single sensor type.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

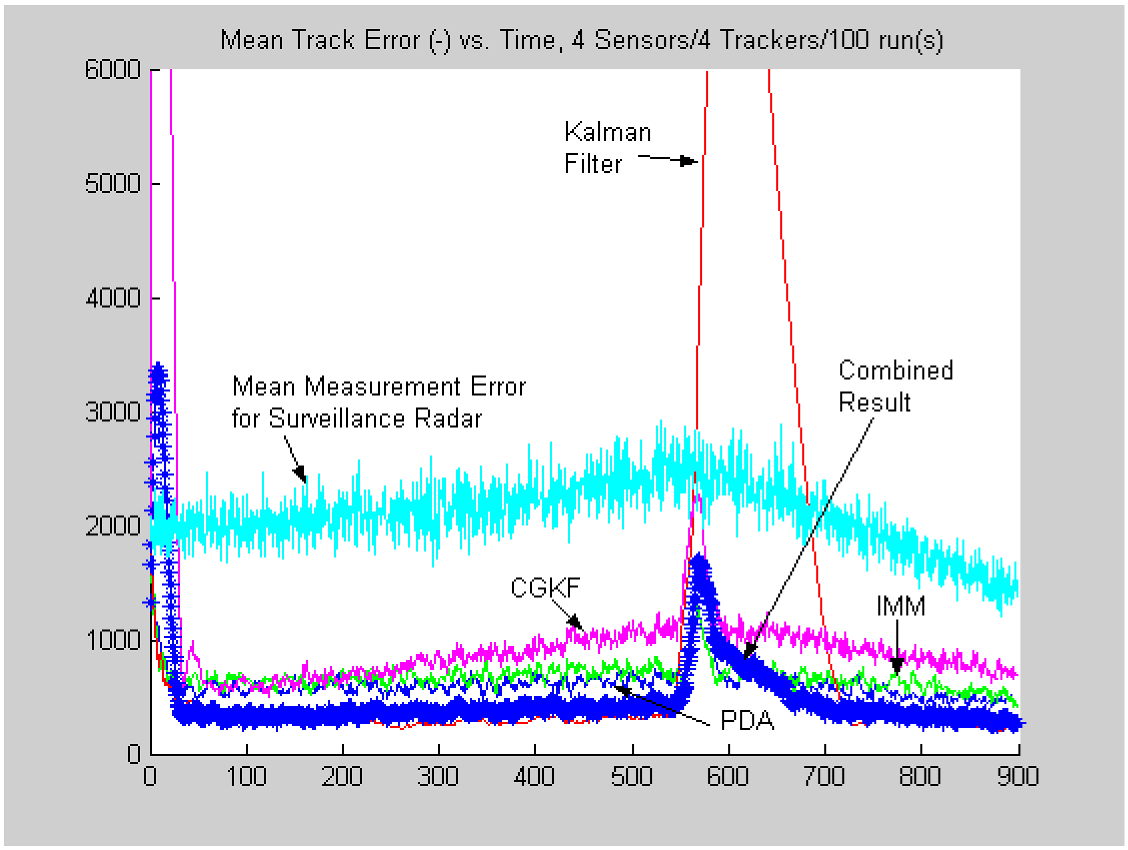

Figure 23. Simple Combined Solution for Maneuvering Target.

Test 3: Set (a)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using a single sensor type.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

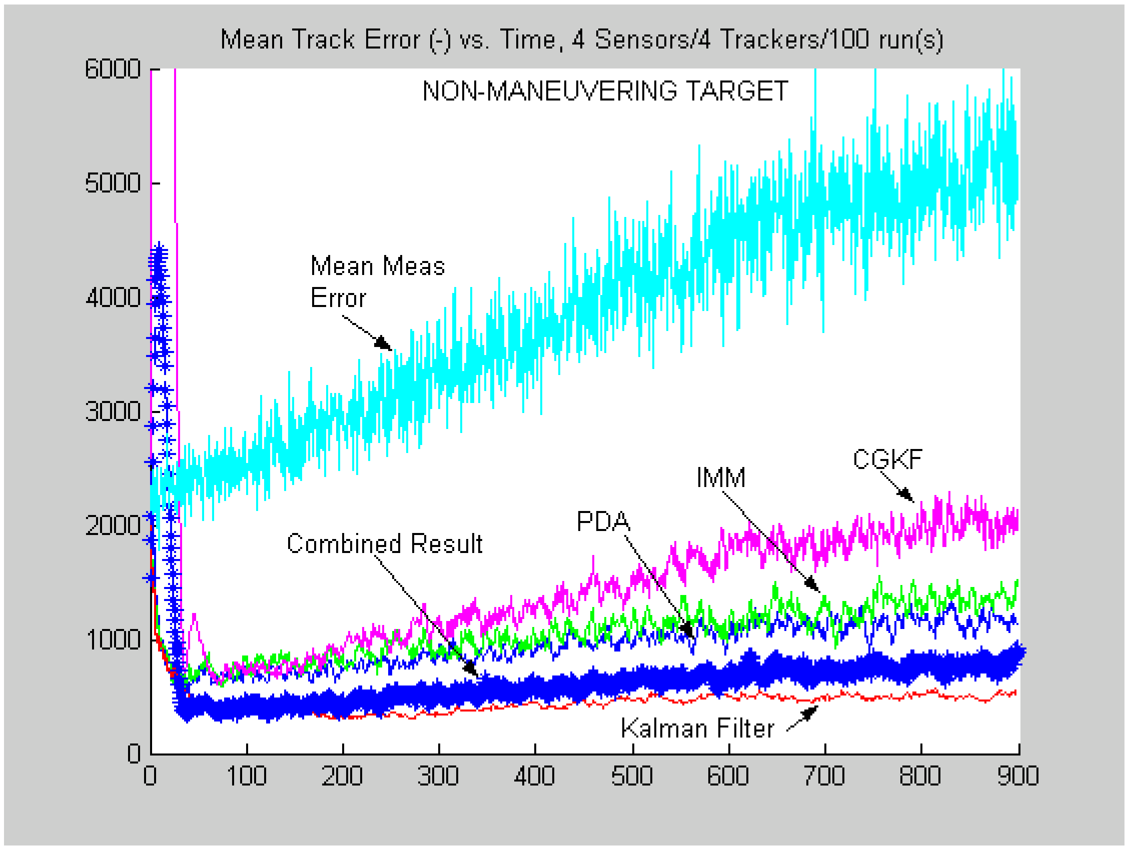

Figure 24. Simple Combined Solution for Non-Maneuvering Target.

Test 3: Set (a)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using a single sensor type.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

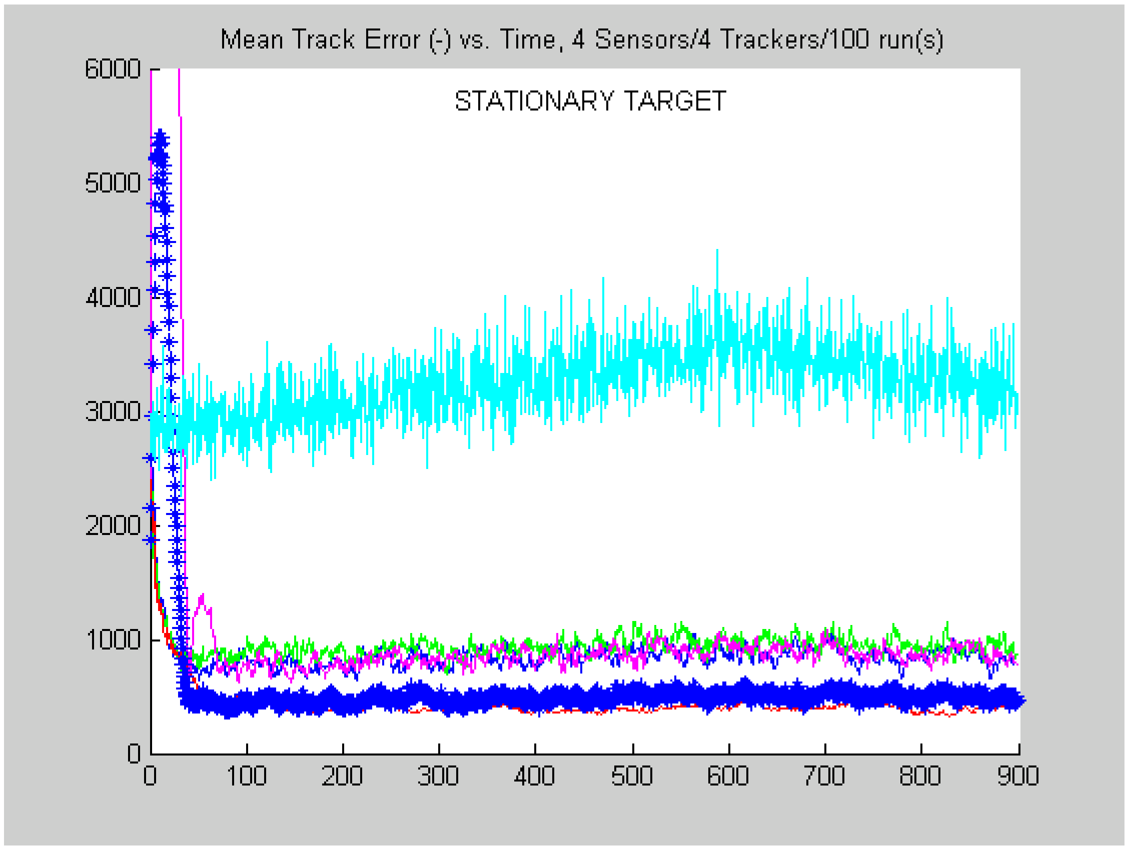

Figure 25. Simple Combined Solution for Fixed Site.

Combined Solution Set (b) Results

For the 2nd set of runs, four different sensor types were used to observe the benefits of combining the data to form a composite tracking solution. Because of the diverse nature of the sensors, this implementation was somewhat representative of an actual sensor platform system. The combined solution was an attempt to take advantage of the best qualities of each of the sensor/tracker pairs. The model has a sensor that can handle clutter, a sensor that was designed to handle turning motion, an optimum straight-line-motion tracker, and a simplified tracker. A simple truth that has been alluded to in previous sections is the nature of the Electronic Surveillance Measures (ESM) and the Infrared (IR) sensors. They are ESM and IR sensors in name only. In fact, the way that the ESM and IR sensors are modeled in the *fusim* program more closely resemble radar or IFF detection systems with greatly reduced resolution. For one other observation, it is unrealistic to expect an ESM system to maintain track on a moving/maneuvering target, since it requires a cooperative target. The ESM solution is for demonstration purposes only.

For this set of runs (Figures 26 through 29), the sensors did not have significantly large errors. However, the plots still show a significant reduction overall for the four sensors. In the turn, only one of the four sensors/tracker pairs reported high errors. Between the two maneuvering target Figures (26 and 27), the differences in the plots show how target orientation with respect to the sensor platform can have an impact on the target errors and the combined solution. The error was significantly reduced by combining the results of the different systems.

As evidenced in all the previous plots (and previous tests), it would be advantageous to disregard the solution obtained with the CGKF until the initial tracking error has settled to below the measurement noise. This is another area where errors in the combined solution could be truncated.

To demonstrate that the combined solution was not terribly susceptible to more realistic errors in the IRST system, the errors were adjusted to observe the effects on the combined solution (Figures 30 through 31).

In the resulting plots of Figures 30 and 31, the IRST/CGKF pair has an even larger initialization error due to track start-up. This unwanted result had an even greater effect on the combined solution, indicating a need to let the initialization errors settle prior to using the data with any confidence.

Test 3: Set (b)	Simple Combined Solution for Multiple Target/Multiple Sensor			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using sensors of varying type			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

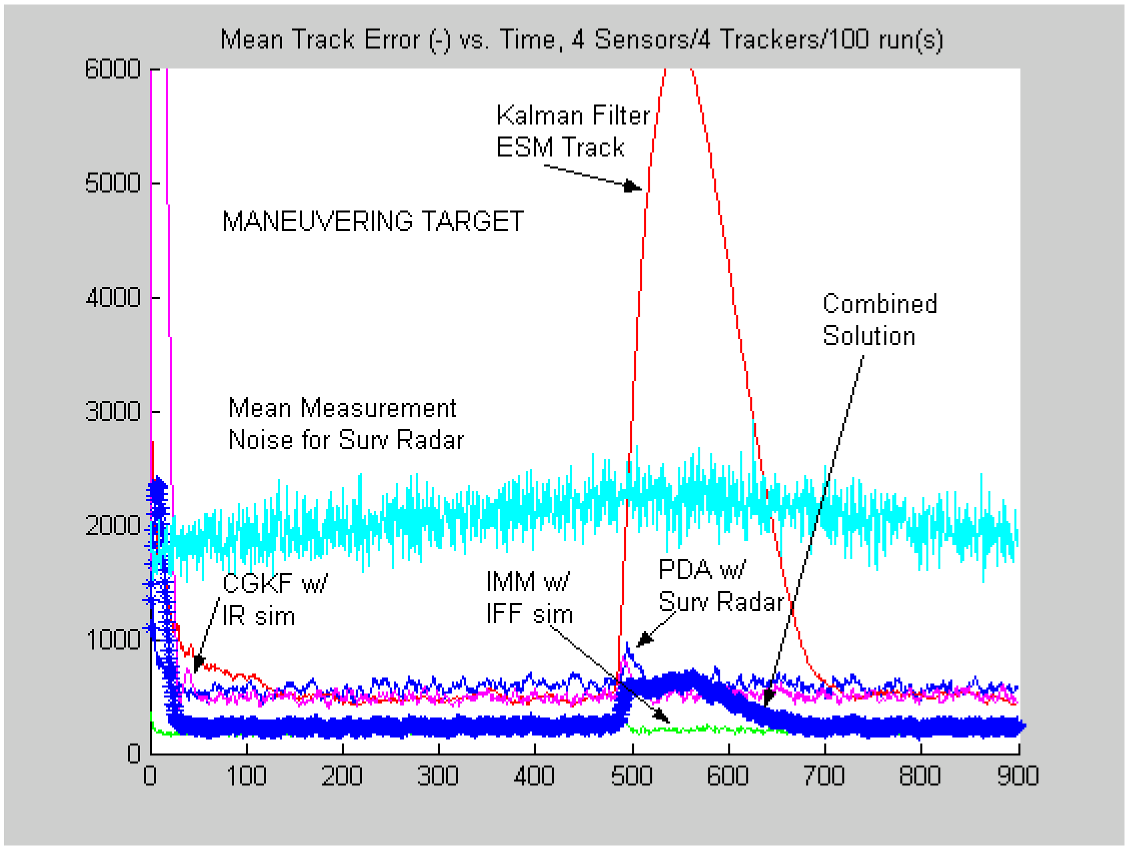

Figure 26. Simple Combined Solution with Diverse Sensors, Maneuvering Target.

Test 3: Set (b)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using sensors of varying type			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

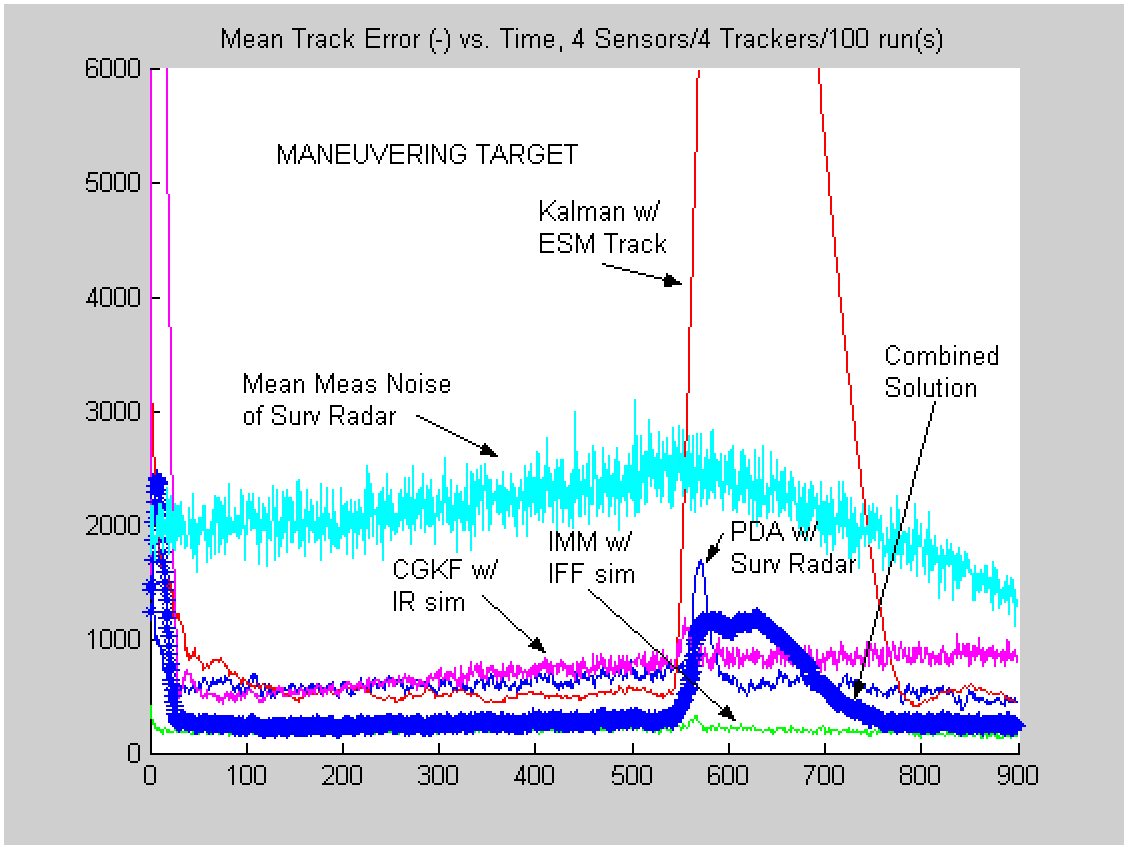

Figure 27. Simple Combined Solution with Diverse Sensors, Maneuvering Target.

Test 3: Set (b)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using sensors of varying type			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

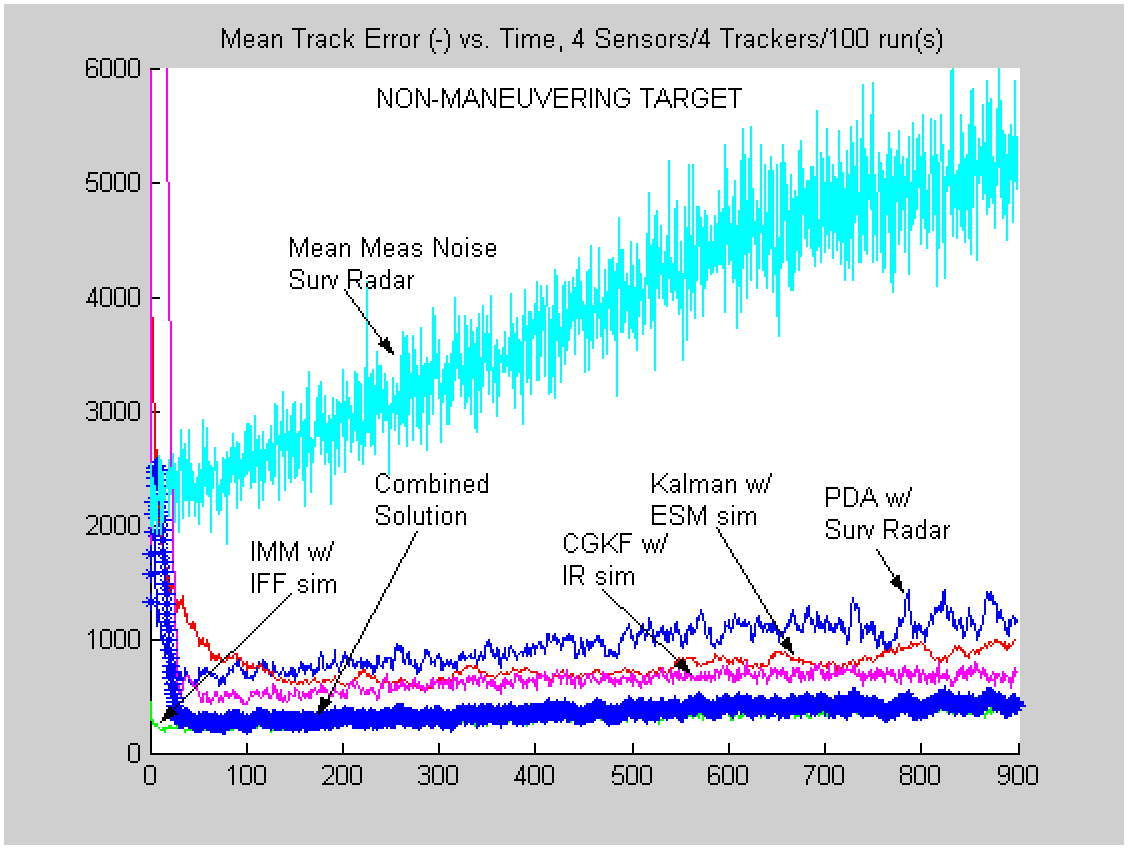

Figure 28. Simple Combined Solution with Diverse Sensors, Non-Maneuvering Target.

Test 3: Set (b)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using sensors of varying type			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

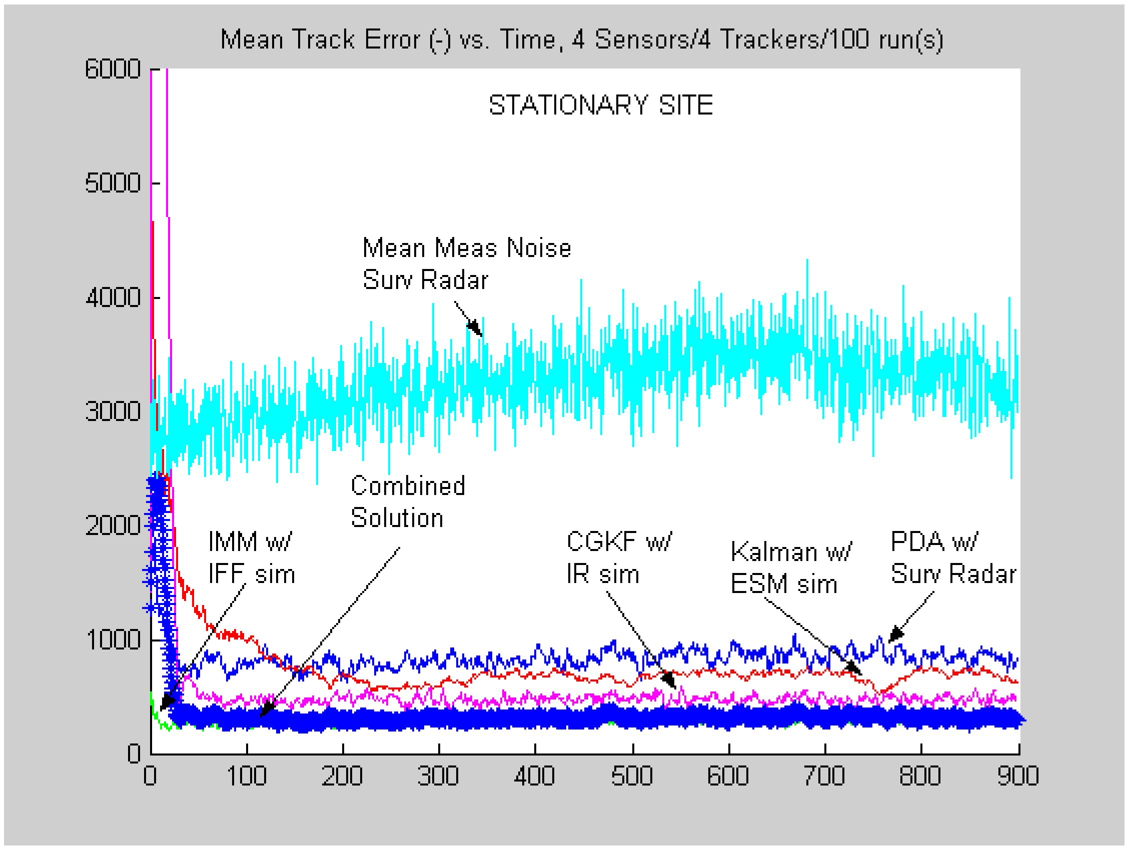

Figure 29. Simple Combined Solution with Diverse Sensors, Fixed Site.

Test 3: Set (b)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using sensors of varying type			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

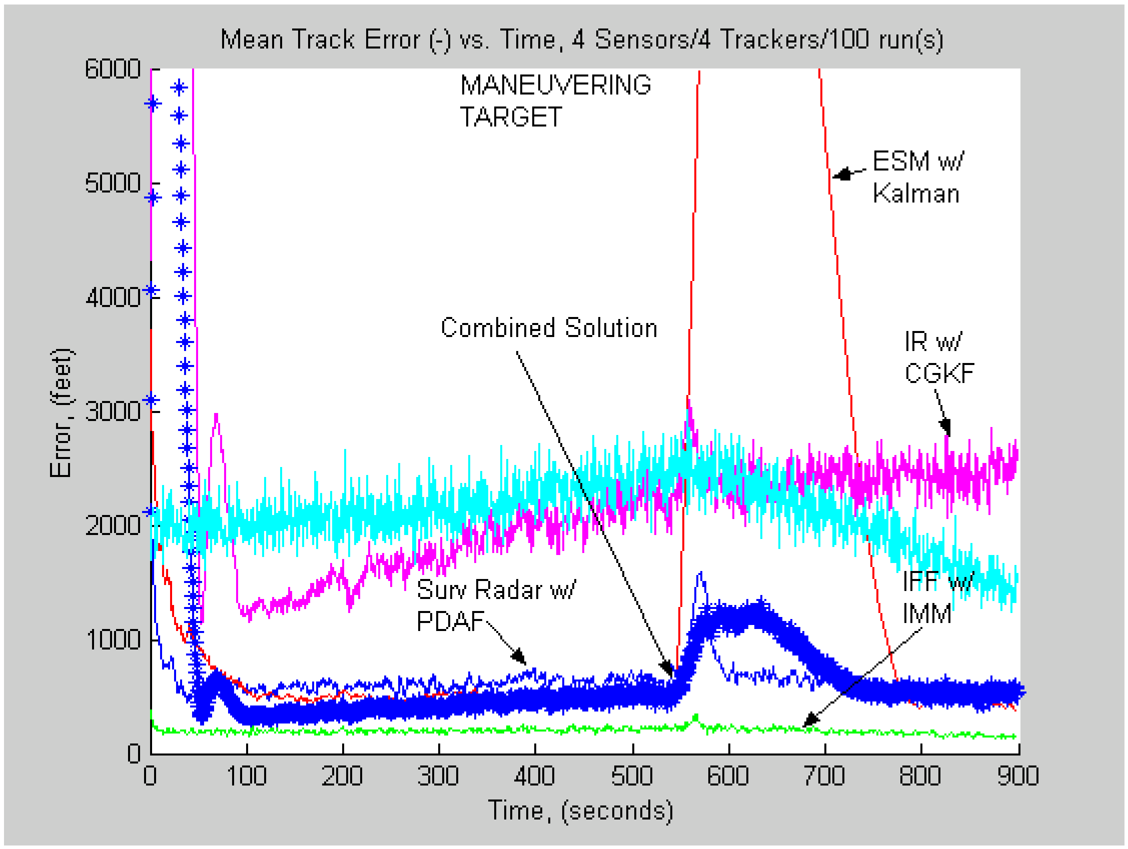

Figure 30. Combined Solution with Diverse Sensors, Altered Sensor Errors.

Test 3: Set (b)	Simple Combined Solution for Multiple Target/Multiple Sensor			
Description:	Demonstrate the results of combining the solutions into a single output state. Demonstrate the performance of the trackers using sensors of varying type			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 1000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 10000.0	- q^2 = 2.0	- q^2 = 1000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

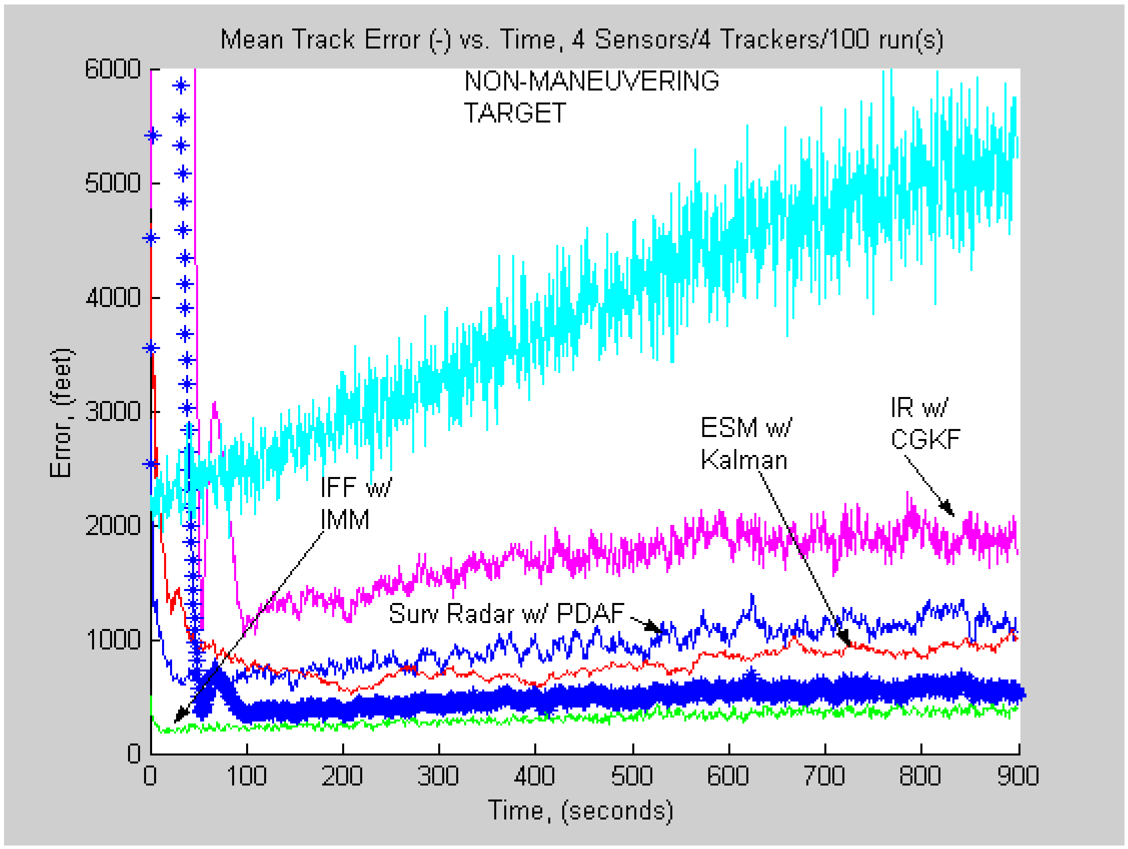

Figure 31. Combined Solution with Diverse Sensors, Altered Sensor Errors.

Combined Solution Set (c) Results

For the third set of runs in the Test 3 configuration, the noise term (q^2) was increased in each of the tracking filters to allow better performance when tracking maneuvering targets. Also, observing the overall effect on the combined solution was an objective of this test set. The tracker parameters were altered as shown in Table 6.

Tracker:	**PDAF**	**IMM**	**KF**	**CGKF**
Previous q^2:	1000	SLM: 1 Turn: 10000	2	1000
Set 3 q^2:	5000	SLM: 1 Turn: 100000	5000	5000

Table 6. Altered Parameters of Combined Solution Set 3.

For this set of runs, the overall combined solution was slightly better than expected. Part of the reason for the success of the solution was the continued effective performance of the IMM. Although the IMM was paired with a more accurate sensor, increasing the turn q^2 did not greatly affect the SLM performance. The Kalman filter solution for the ESM system showed the most dramatic increase in overall error (figure 32), while the CGKF solution for the IRST system was only slightly increased. However, the turn performance in each of the trackers improved, resulting in virtually no increase in error due to turning motion in the combined solution.

Against the non-maneuvering target (Figure 33), the ESM/Kalman filter solution was no better than the IRST/constant gain case, even though the IRST sensor had much more error than the ESM system. Part of this poor performance in the KF can be attributed to a very large distance from the sensor platform. However, when compared to the low noise case (q^2 = 2) of figure 31, the KF did a fine job of tracking over great distances even when the sensor error was fairly high (as in the case of the ESM system). The decreased performance of the Kalman filter from Figure 31 to Figure 33 is directly attributable to the increased noise term. The Kalman filter also had poor performance against the fixed site in Figure 34. The poor performance of the KF is an indication that a compromise is necessary when using a standard Kalman filter for tracking. Should the Kalman filter be optimized for SLM performance, or for turning performance? The Kalman solution with a low noise term is the optimum solution when the target is not maneuvering. In Test 3a, the Kalman filter had the least error against the non-maneuvering target and the fixed site (a same sensor comparison).

Test 3: Set (c)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining variable solutions into a single output state. Demonstrate the performance of the trackers with increased noise terms.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 5000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 100000.0	- q^2 = 5000.0	- q^2 = 5000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

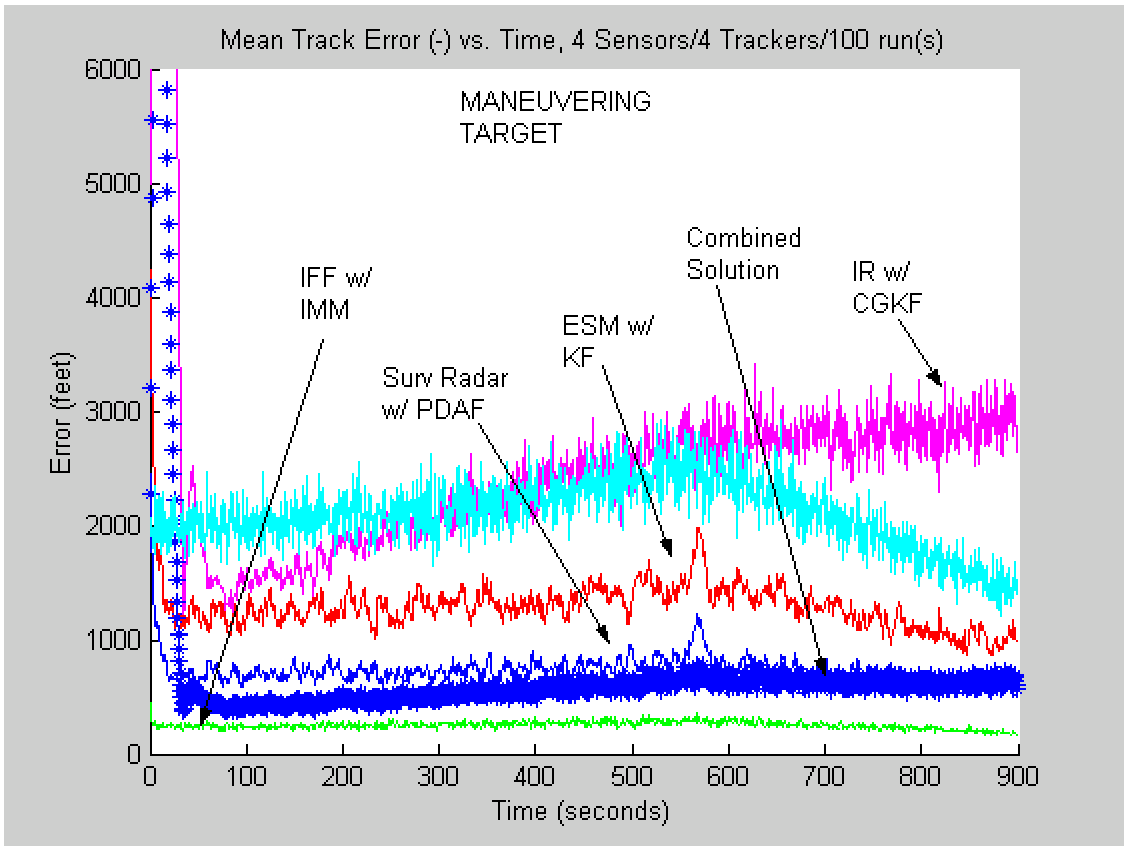

Figure 32. Combined Solution with Diverse Sensors, Altered Tracker Parameters.

Test 3: Set (c)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining variable solutions into a single output state. Demonstrate the performance of the trackers with increased noise terms.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 5000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 100000.0	- q^2 = 5000.0	- q^2 = 5000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

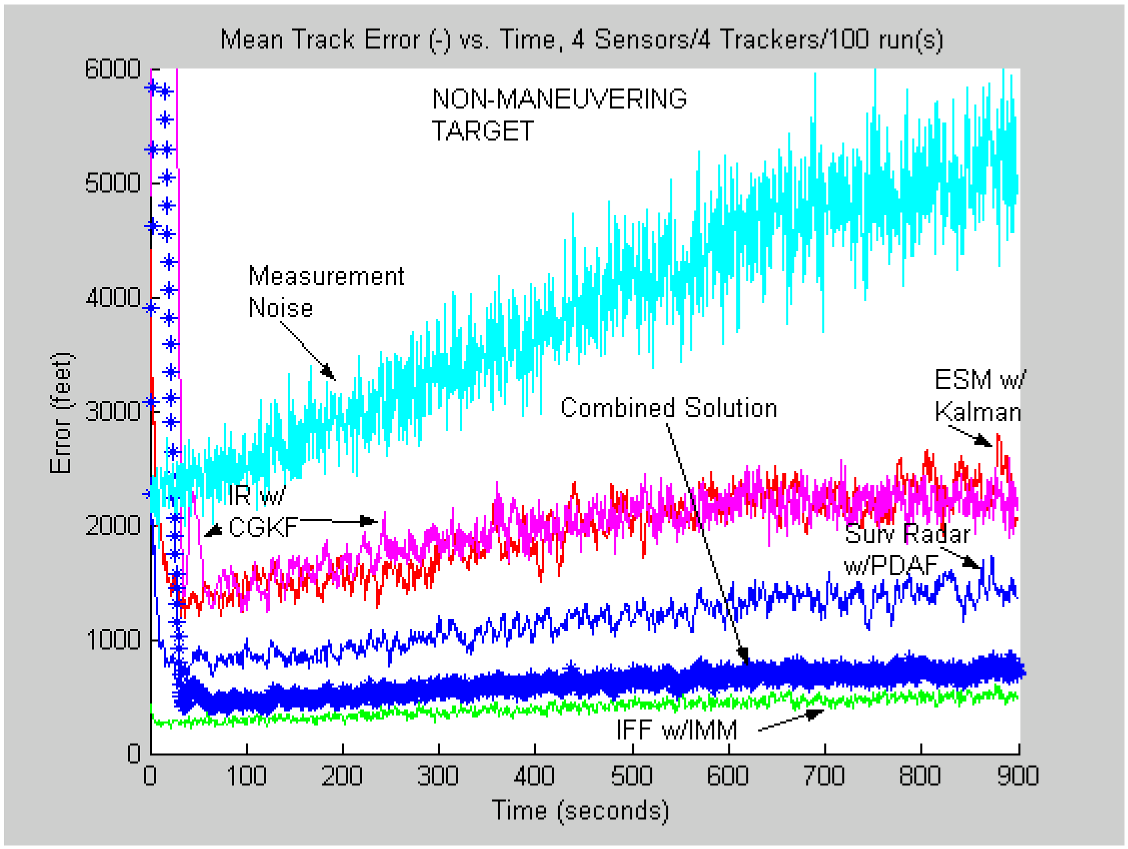

Figure 33. Combined Solution with Diverse Sensors, Altered Tracker Parameters.

Test 3: Set (c)	**Simple Combined Solution for Multiple Target/Multiple Sensor**			
Description:	Demonstrate the results of combining variable solutions into a single output state. Demonstrate the performance of the trackers with increased noise terms.			
Sensor	Airborne Surveillance Radar	Identification Friend or Foe System	Electronic Surveillance System	Infrared Search and Track System
Tracker	PDAF	IMM	Kalman Filter	CGKF
Settings:	- q^2 = 5000.0 - 1 clutter pt	- SLM q^2 = 1.0 - Turn q^2 = 100000.0	- q^2 = 5000.0	- q^2 = 5000.0
Targets	Maneuvering Aircraft	Non-Maneuvering Aircraft	Slow-Moving Ship	Fixed Site

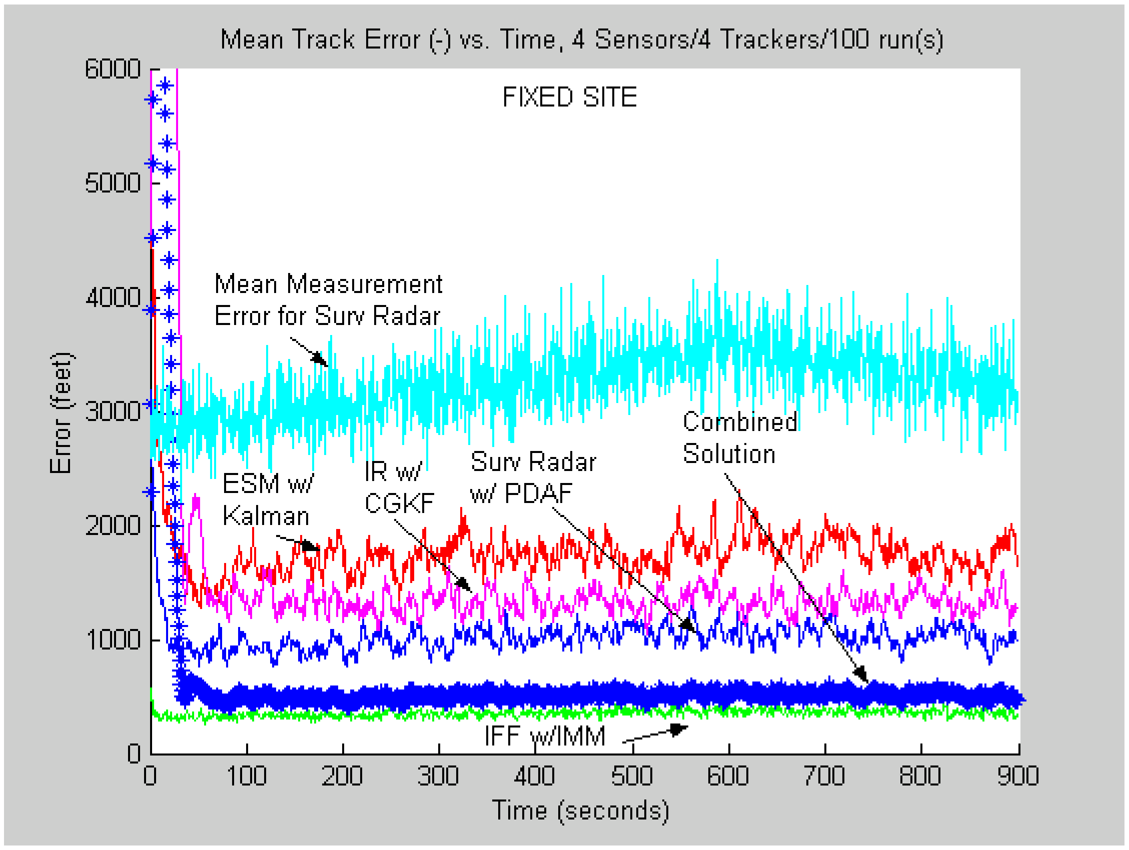

Figure 34. Combined Solution with Diverse Sensors, Altered Tracker Parameters.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

Testing algorithms in a simulation environment can be an effective way to study the sensor fusion problem without using live sensors and targets. The cost of trying to use actual targets with repeated initial conditions would be astronomical. A simulation scheme can test and retest an algorithm using randomly generated perturbations to provide an indication of the algorithm effectiveness. The *fusim* program was written to give the user flexibility in selecting a sensor platform, the sensors used on the platform, the target types, the problem orientation, and the tracking algorithms to be used with the sensors. To change parameters, the user can simply alter the MATLAB code within the applicable files to produce the desired output. The *fusim* program can be used to compare tracking algorithms in a multiple sensor/multiple target environment, view the effects of using diverse sensors, and evaluate the effects of data association algorithms. For this early edition of the *fusim* program, only a rudimentary combination of sensor/tracker outputs was used to evaluate the effects of multiple, diverse sensors.

The *fusim* program was designed to allow averaging over numerous runs to find a mean solution for each sensor and track update. The error generation was based on Gaussian distributed random white noise, meaning that numerous runs and average solutions would be necessary to account for the randomness of the noise. For most test cases, 100 or 200 runs were used to produce a smooth track output and error plot. However, this information tells nothing of the tracker performance for a single run. Therefore, the program also allows plotting results from a single run. Data gathered over a single run may be more relevent for evaluating a tracking or fusion algorithm once the algorithm is proven. When used against live targets, the tracker will have to perform consistently and predictably.

In this study, the effect of multiple sensors in different locations contributing to the tracking picture was not analyzed. None of the track results were able to take advantage of multiple sensor locations, as in the case of a Cooperative Engagement Capability system with multiple participating units. Also, none of the tracking algorithms were true multiple sensor/multiple track algorithms. Throughout the simulation, every measurement was assumed to be associated with only one track. To create a true multiple sensor tracking routine would require measurement-to-track and track-to-track data association. Handling of multiple tracks would require the use of a Multiple Hypothesis Tracker or a Joint Probabilistic Data Association Filter, or some other means of measurement-to-track data association.

The performance of each of the trackers was highly dependent on the errors within each sensor, and in the way that each sensor measured target data. One of the artificial aspects of the *fusim* program is that all sensors report measurements in range and bearing. For example, the modeling of the ESM and IR sensors was greatly oversimplified in the *fusim* program to allow simple application of the measurement and tracking functions. Instead of using bearing only, such as in the case of an ESM target, error was added to an x-y position report. For all sensors in the *fusim* program, errors were modeled in range and bearing. The true x-y position reports were converted to range and bearing, the range/bearing errors were added, the position report was converted back to x-y coordinates, and the 'noisy' measurement (and error covarinance) was returned to the main program for use in the tracking functions. This same procedure occurred for all sensors and tracks regardless of type. In Test 1 Set (a), where the sensor errors were zero, the tracker that was used was irrelevant for all cases except the maneuvering target. Some error was noted in the turn, suggesting that the kinematics of the problem has an effect on the target state even if the errors in the sensor are zero. Also, nearly every test case involved a moving sensor platform. The sensor platform implementation was a little more realistic, but the analysis was more challenging since the orientation of the problem changed continuously. The data is not as smooth as it would be for a fixed sensor platform with no navigation errors in the position. This analysis provided a general idea of how each of the trackers performed from the perspective of the sensor platform.

The *fusim* program was used to compare the performance of several different tracking algorithms. The PDAF was the only tracker in this evaluation that was forced to deal with clutter. Because of the clutter variable, the PDAF was not as consistent and predictable as the other trackers. On one set of runs, the PDAF ran away from the actual position of the stationary target due to a slight velocity induced in the state vector by the clutter point. Since the clutter point was placed around the predicted state and the tracker does not distinguish between the clutter measurement and the actual measurement, the state was allowed to move away from the actual position of the target. The data link report followed suit, and provided a good illustration of how easy it is to flood a system with bad information. The information output from a system is only as good as the information input.

The IMM was the most consistent tracker of the evaluation set for maneuvering targets. However, the IMM also performed fairly well with non-maneuvering targets, slow-moving targets, and fixed sites. In the algorithm, the only difference between the two models was the noise factor q^2 in the SLM model and the turn model. The IMM produced the best overall tracking solution every time, but required much more code and computation time (see Appendix B).

The Kalman filter performed very well when tracking non-maneuvering, slow-moving, and fixed-site targets. The Kalman filter did not handle maneuvering targets very well. By increasing the plant noise, the Kalman filter handled the turns better, but resulted in noisier straight-line tracking with a higher average error.

Repeatedly, the Constant Gain Kalman Filter produced reasonable straight-line performance, but did not do as well during initialization or in turning motion. The tracker required the fewest computations, but the large track initiation errors were present in every test case, except for the zero-error case. However, when the measurement errors were low due to sensor accuracy or due to the target position relative to the sensor platform, the CGKF performed as well as the PDAF or the IMM for the same sensor/target pair. This suggests that the CGKF would be useful in high-workload situations where non-priority tracks could be tracked with less computational strain, or the CGKF would be useful when sensor errors are low.

The applications of sensor fusion are widely varied, but particular attention is paid to the military application in this thesis. Battle group operations, the Common Operational Picture, land attack scenarios, time-critical strike, and satellite data applications are just a few examples of the military uses of Sensor Fusion. The ‘Sensor Fusion’ problem is very broad in scope, to include network centric interconnectivity, platform level processing and display, and global adaptivity for flexible mission application. This thesis does not address the global aspects of sensor fusion such as interoperability or systems engineering, nor does it address platform level systems engineering. Rather, this thesis was an attempt to reach into the core of sensor fusion algorithms and discuss the tracking and data association aspects of tracking algorithms.

The *fusim* program was designed with a great deal of flexibility to allow for future applications and upgrades. It is recommended that this study be continued to allow the following expansions:

1. Extend this 2-dimensional study to 3-D with full coordinate system accounting.
2. Produce functions that allow full data association for (1) measurement-to-track (2) track-to-track (3) data-to-track.
3. Integrate an emitter library, similar to the sensor file, that could be used to provide more realistic ESM system inputs, and attribute information.
4. Expand the program to include full accounting of track attributes for determining track identification, and allow attribute tracking.
5. Design the Graphical User Interface for maximum user benefit of fusim and study the Human-Machine Interface aspects of Sensor Fusion.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Bar-Shalom, Y., and Li, X-R, *Multitarget-Multisensor Tracking: Principles and Techniques*, Third Printing, 1995.

2. Waltz, Edward and Llinas, James, *Multisensor Data Fusion*, Artech House Inc.,1990.

3. Steinberg, Alan N., and Bowman, Christopher L., and White, Franklin E., " Revisions to the JDL Data Fusion Model", paper presented at the Joint NATO/IRIS Conference, Quebec, October 1998.

4. *Multi-Source Integration Systems Engineering Team (MSISET) Final Report to the Combat Systems Functional Allocation Board*, PowerPoint Presentation, Unpublished, 1997.

5. Hinman, Michael L., *Situation Assessment and Threat Prediction*, PowerPoint Presentation, Air Force Research Lab, Unpublished, March 2001.

6. Air Force Research Laboratory Final Technical Report, AFRL-IF-RS-TR-2000-99, *Threat Prediction Fusion*, by Ross, Kenneth N., and Chaney, Ronald D., July 2000.

7. Air Force Research Laboratory Final Technical Report, AFRL-IF-RS-TR-1999-74, *Adaptive Data Fusion Technology*, by Ross, Kenneth N., and Chaney, Ronald D., April 1999.

8. Blackman, Samuel; Popoli, Robert, *Design and Analysis of Modern Tracking Systems*, Artech House, 1999.

9. Hutchins, Robert G., *Optimal Estimation, Kalman Filters and Target Tracking*, Class Notes, Naval Postgraduate School, July 2000.

10. Air Force Research Laboratory Final Technical Report, AFRL-IF-RS-TR-1998-58, *AWACS Track and Identification Fusion*, by Collins, Noel, April 1998.

11. Drummond, Oliver E., "Integration of Features and Attributes into Target Tracking", for the Office of Naval Research, Unpublished, 2000.

12. Bar-Shalom, Y., Blair, W.D., *Multitarget-Multisensor Tracking: Applications and Advances*, Volume III, Artech House, Inc., 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

Bar-Shalom, Y., Li, X-R, *Estimation and Tracking: Principles, Techniques and Software*, Artech House Inc., 1993.

Bar-Shalom, Y., *Multitarget-Multisensor Tracking: Applications and Advances*, Volume I, Artech House, Inc. 1990.

Bar-Shalom, Y., *Multitarget-Multisensor Tracking: Applications and Advances*, Volume II, Artech House, Inc., 1996.

Office of Naval Research, Georgia Tech Research Institute, *Third ONR/GTRI Workshop on Target Tracking and Sensor Fusion,* Workshop Notes, May 2000.

Stone, L.D., Barlow, C.A., Corwin, T.L., *Bayesian Target Tracking*, Artech House Inc., 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. *FUSIM* FUNCTIONAL DESCRIPTION

Fusim Functional Description

fusim

The FUSIM file is the main control file for the entire simulation. FUSIM calls the various associated functions to set up the needed matrices and variables, creates the truth data for the targets, adds the measurement error, sends the noisy measurements to the tracker, and then stores and plots the target states and associated least square errors. The FUSIM function is called from the user interface or from the MATLAB command line. The MATLAB code for all the functions is included in Appendix B. First, the START and FTURN functions are called to initialize all the timing and target parameters, as input by the user. From this, the total number of maneuvering and non-maneuvering targets are determined and the sensor information (including number and types of sensors) is extracted. Based on the user selection, the primary tracker type is extracted for the primary sensor, and the other sensors are assigned the remaining trackers. This information is stored in the (trkr) vector.

Next, the outer loop for the Monte Carlo runs is set up, and all applicable variables and matrices become initialized. This way, for repeated runs, the variables are reset to null or to their initial values at each iteration of the loop. The generation of truth data begins with extracting the initial positions using the NONMAN and MANMATRIX functions. The maneuvering and non-maneuvering targets are kept separate for generating the truth data, and then regrouped for generating the noisy measurements. The initial position and velocity vectors look like [x1;vx1;y1;vy1, x2;vx2;y2;vy2, ...]

for each target. In order to generate the second measurement, the TIMESTEP function is called to iterate the target motion in time.

The MANMATRIX function has an additional task that is not part of the NONMAN function. MANMATRIX also returns the (mantime) matrix that contains the column vectors of maneuver times for the maneuvering targets only. However, for the first two measurements, straight line motion for all targets is assumed. The truth data is then stored in the (posout) matrix as x-y pairs stacked for each target. A new column of x-y pairs for each target is created with each time step.

The first two measurements are required for track initialization. This is a major assumption of this simulation. All tracks are initialized with a Least Squares Estimation track initialization, regardless of the tracker used in the simulation. Trackers that use Kalman predictions and updates require an initial state and covariance for the tracker to work. The INIT function is called for each sensor of the sensor platform. This is where the multi-dimensional capability of MATLAB is first exercised. While the 2-D form of the matrix is needed for the calculations, storage of the multiple sensor data is accomplished by adding a “page” to the 2-D matrix for each additional sensor. This gives the matrices depth, or a third dimension. Storing this data separately is necessary because each sensor has a unique accuracy. The SETZTRUE function is used to conveniently stack the z-true data for storage and plotting. The noisy measurements are also saved for error comparisons after the tracker has updated the state (zout).

At this point, the simulation is completely initialized and ready to start looping through the main body of the program. For each iteration, the following actions occur:

1. Truth data is generated for non-maneuvering targets with TIMESTEP.

2. A test is performed for each maneuvering target to see if the target is in straight line motion or in a maneuvering state. If the target is in SLM, the straight line (F) matrix is sent to TIMESTEP. If the target is in a tuning state (based on 'mantime' matrix times), the (Fturn) matrix is sent to TIMESTEP for proper modeling of the turn.

3. All of the truth data is combined and sent to the MEAS function for generation of noisy measurement data. The error and the error covariance is returned for each target and for each sensor, which add to the 3-D matrices already generated by INIT.

4. The noisy measurement data, error covariance, initial state and initial state covariance are all sent to the applicable tracker for each sensor. This data is all kept separate for plotting the individual solutions and errors.

5. Once all the data and tracking solutions have been calculated for the entire simulation run, the LSECALC function determines the Least Square Error between the tracker state and the truth position of each point and for each sensor. If multiple Monte Carlo runs are conducted, the results are stored and averaged over the number of runs. This averaged data is used for plotting the final results. An additional option is allowed in the code for plotting the state solutions at each loop iteration (SIMPLOT). The functions TRACKPLOT and ERRORPLOTS are used to plot the results.

When running *fusim*, the track plot will look similar to the results in Figure 35.

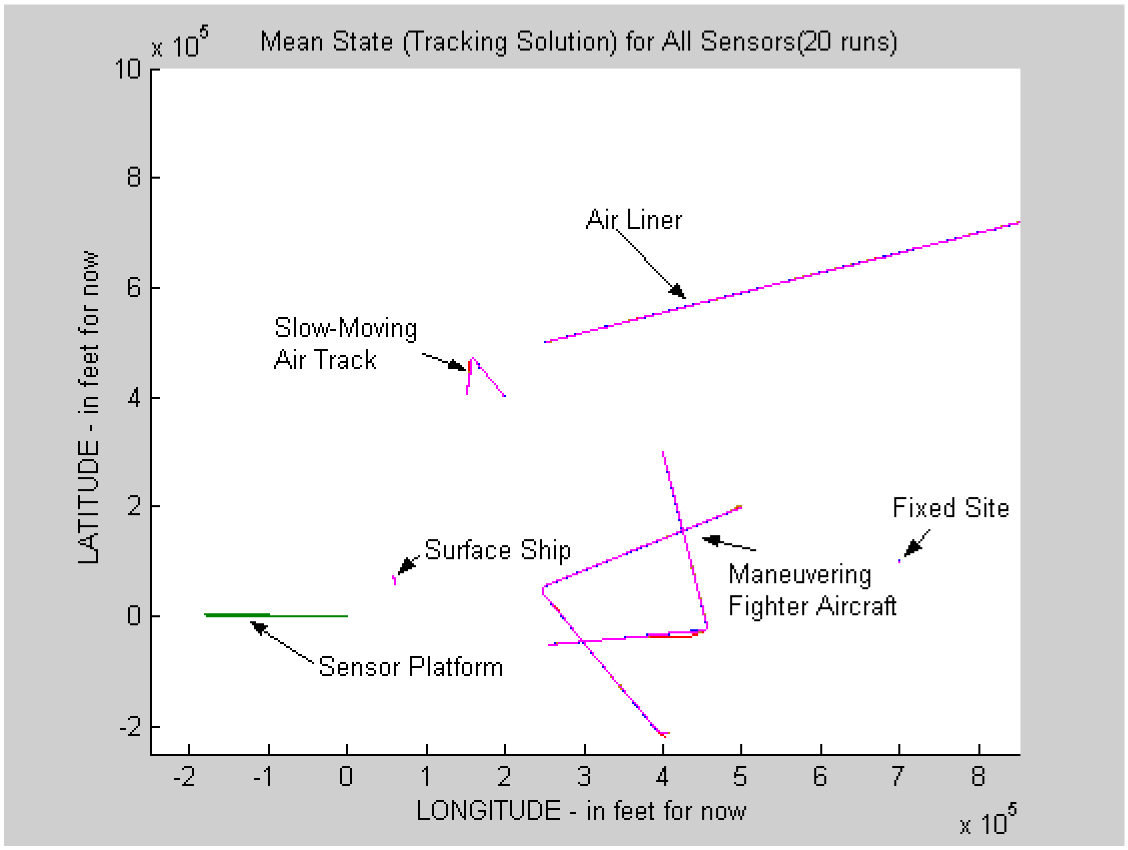

Figure 35. Representative PPI.

[A,A1,delta,t,nloops,nsamples,Spi,flag,ownsens] = start(dumb);

The START function is called first by FUSIM to set up the initial variables needed for the simulation. The START function reads the START.MAT file to collect timing information about the simulation, initiates the matrices of targets, and initiates other needed matrices (F,H). The notable functions and variables of the start function are described in the following sections.

The START function begins by reading the START.MAT file to initialize the timing variables and other user-defined variables. The START.MAT information initializes variables of the (simstart) vector in the following format: [t(simtime), nloops, deltatime, DLRP, placeholder, tracker selection 1-4]. The total simulation time (t) is input in minutes and then converted to seconds. The number of loops (nloops) determines the total number of Monte Carlo runs to be executed by the program. A large number of runs can add significantly to the execution time required by the simulation.

The time variable (deltatime) is the update rate of the sensors given in tenths of minutes. For example, if 0.1 is used, 0.1*60 = 6 seconds. In other words, the update rate of the primary sensor is every 6 seconds. The variable (DLRP) is used to center the entire simulation at some location on the earth. When entered in the simulation, (DLRP) becomes the origin of the X-Y coordinate grid. Note: The (DLRP) variable is not used in this version of *fusim*. The tracker selection is used in a later function. Now that the total simulation time and the time step (delta) are known, the total number of samples (nsamples) can be calculated. The variable (nsamples) is an integer value that provides FUSIM with the total number of samples or measurements to be taken for each of the targets.

Next, the TARDAT.MAT file is read into the (A1) matrix. By examining the elements of the (A1) matrix, a determination is made whether the target is maneuvering or not. This information will be used later for setting up the truth data. As the maneuver test is performed, the targets are separated into the (A) matrix (maneuvering targets) and the (A1) matrix (non-maneuvering targets). These two matrices contain all the truth data about all the targets as input by the user. The truth elements of the sensor platform and of the target vectors are nearly the same, as shown below and further described in Appendix B:

Example sensor platform vector (format of TARDAT.MAT data):

[N,365600,W,0761700,110,175,25000,A/C,e2c,radar1,iff,esm,ir,comm,legtime,L/R,turn time]

1: North or South Latitude N = 0, S = 1 (not utilized in this version)

2: X position, with 0/0 as the origin (feet)

3: East or West Longitude: W = 0, E = 1 (not utilized in this version)

4: Y position, with 0/0 as the origin (feet)

5: Course in degrees

6: Speed in Knots

7: Altitude in feet (not utilized in this version)

8: Type: 1 = Aircraft, 2 = ship, 3 = ? (not utilized in this version)

9: Name 1 = E-2C, 2 = ? (not utilized in this version)

10: Sensor 1 (sensor platform) / Emitter 1 (target)

11: Sensor 2 (sensor platform) / Emitter 2 (target)

12: Sensor 3 (sensor platform) / Emitter 3 (target)

13: Sensor 4 (sensor platform) / Emitter 4 (target)

14: communications type (not utilized in this version)

15: Leg Time - how long the legs of the orbit will be (if turning)

16: Turns - Left or Right, 0 = Left Turn, 1 = Right Turn

17: Turn Time - how long the turns will be (if turning)

Three other functions are initialized for the sensor platform. The sensor platform position (Spi) is initialized for use later in the program, flag is set to 1 if the sensor platform is maneuvering, and the vector of sensor types (ownsens) is initialized with the user-defined selections of sensor types. Up to 4 different sensors can be selected for the sensor platform.

Two other matrices are initialized in the START function, the (H) matrix and the (F) matrix. These are declared as global variables because they are used by several other functions in the FUSIM program. The (H) matrix is used simply to extract the x-y position variables from the state vector. The (F) matrix is used to increment in time a state vector for predictions and for establishing truth data when targets are in straight-line motion.

[Fturn,B] = fturn(A,delta);

The function FTURN is called next by the FUSIM control function to set up the needed matrices for maneuvering targets. An fturn matrix is used by each maneuvering target to model the target motion when the target is in a turning state. The full (Fturn) matrix is returned with the stacked 4x4 (Fturn) sub-matrices for the maneuvering targets only. [Fturn(ownship);Fturn(target1);Fturn(target2);....]. The (B) matrix is set up by determining whether the target is turning left or right, extracting and converting the heading and speed information, and the initial truth state vector for each maneuvering target. The data for target 1 (usually ownship) occupies column 1, target 2 is in column 2 and so on for the total number of maneuvering targets.

function trkr = setuptrackers(ns)

The function SETUPTRACKER uses the input value (ns) to set up a vector of sensor types. The value (ns) is based on the number of sensor types selected by the user.

The only tracker that the user can select is the primary tracking function desired for the primary sensor. The rest of the sensors are arbitrarily assigned a tracking function. Currently, the trackers are assigned as follows:

1 =	Probabilistic Data Association Filter
2 =	Interacting multiple Models
3 =	Kalman
4 =	Const Gain Kalman

function [mantime,xim] = manmatrix(A,B,t)

The function MANMATRIX constructs the Maneuver Time Matrix based on the size of the largest maneuver vector. First, the function extracts the maneuver times from the (A) matrix and converts the user-defined times to seconds. Using the leg and turn times, the maneuver vectors are then created with the MANEUVERTIME function, as discussed in the next section. The maneuver times for target 1 occupies the first column, the times for target 2 occupy the second column, and so on. If the vectors are different in length, the number of rows in the maneuver matrix is based on the longest vector. The rest of the matrix is filled in with zeros. The MANMATRIX function also returns the initial truth position (xim) for the maneuvering targets.

function [mt] = maneuvertime(t,legt,turnt)

The function MANEUVERTIME is called for each target that is maneuvering. Based on the total time of the simulation and the leg and turn times of the target, the function creates a vector of cumulative maneuver times. The first time element in the vector will always be the first leg time. The next time will be the length of time required to complete the first turn added to the first leg time. The target uses the straight-line motion model (F) for the first leg, and the turning model (Fturn) during the time that the target is turning. After completion of the turn, the target returns to SLM until the next turn time is reached. As an example, if the first leg is 600 seconds, the target will use SLM model for 600 seconds. If the turn takes 30 seconds to complete, the turn model will be used from time 601 to time 630. At time 631, the target returns to SLM until time 631 + 600 or 1231 seconds. This target motion model continues until the total time of the simulation is reached.

function [xin] = nonman(A1)

The non-maneuvering targets are initialized using the NONMAN function. This function extracts the position and velocity data from the (A1) matrix of non-maneuvering targets, converts the speed to feet per second, and breaks down the velocity into x and y components based on the heading. This also includes fixed targets that have no motion associated with them. The state vector is constructed for each target and stored as a column of the non-maneuvering matrix.

function [spxi] = ownxi(ownship)

This function simply extracts the initial position and velocity data from the ownship initial parameters vector (as input by the user).

function [Q,P,R,derror,xhat,zret] = init(allx1,allx2,delta,Spi,Sp,sensor)

The INIT function is one of the most important functions in the simulation. This function performs nearly all the same operations as the main FUSIM loop. Since all the trackers used in this simulation require some form of initialization, a simple Least Squares Estimation initialization is performed for each. Using the least squares initialization for all the trackers is a bit of a simplification, but the simulation is much more flexible. Nearly any tracker that uses a Kalman type prediction and update can be plugged into this simulation and evaluated. The INIT function uses the first two noisy measurements (assuming SLM) to initialize the Kalman filter state and the prediction covariance. Using the initial truth data provided by the user, the function POLE2CART is called with the applicable sensor range and bearing error, and the error covariance matrix. Returned from POLE2CART are the noisy measurement, the measurement error, and the new error covariance (based on the measurement). More information on POLE2CART is provided in the following section.

Since two measurements are required for the initialization, the targets are stepped through time for one time increment using the TIMESTEP function. The initialization assumes SLM for all targets. The truth data is stored for later output, and POLE2CART is called again to create a noisy measurement for each target based on the sensor errors. With the 2 noisy measurements, sufficient data exists for the least squares initialization. The data that are returned include the noisy measurements (zret), the initial states (xhat), the error covariance (R), the prediction covariance (P), the measurement errors (derror), and the plant noise (Q) matrix.

function [z,R,disterror] = pole2cart(ztrue,Sp,sr,sb,Sv)

The POLE2CART function simulates the errors inherent within a sensor by adding error to the truth position and returning a new noisy measurement. First, the x-y truth data is converted to polar coordinates based on the sensor position and the target position. Next, the range and bearing errors are multiplied by a randomly generated number to simulate Gaussian distributed white noise, and then added to the true range and bearing. The new range and bearing are then converted back to Cartesian coordinates and the error covariance matrix is constructed. The error between the noisy measurement and the truth position is measured and returned with the noisy position and error covariance matrix. Using POLE2CART for the measurements creates a slight bias error that is ignored for this simulation [Ref. 1].

function [spP,spR,sperror,spstate,spmeas] = spinit(spxi,spx2,Q,D)

The SPINIT function provides the initial gain and state estimation for ownship navigation, assuming straight line motion. The initialization is precisely the same as the target track initializations.

function ztrue = setztrue(m,n,xin,xim)

The SETZTRUE function extracts the true x-y data from the truth state vector. This is performed for both maneuvering and non-maneuvering targets. The data is stored in the (ztrue) vector and returned to the FUSIM function for storage in the posout matrix.

function [sigr,sigb,Sv] = senserror(sensor)

The SENSERROR function matches the type of sensor defined by the user for the sensor platform to the corresponding errors. For each sensor on the sensor platform, the range and bearing errors are returned along with the sensor variance matrix. For simplicity, all sensors are treated the same. The errors are given in range and bearing, allowing use of the POLE2CART function for all sensor measurements. This is not the most accurate representation of sensors, however, treating the sensors the same allows more flexibility in the simulation. The following list describes the sensor type and the errors used for the simulation:

Case	Sensor Type	? $_{range}$, ft	? $_{bearing}$, rad	Sensor Variance
1	Airborne Surveillance Radar	100	.005	?? ${}^{2}_{range}$ 0 ? ? ? ? 0 ? ${}^{2}_{bearing}$? ?
2	Surface Surveillance Radar	20	.0001	same as above
3	Airborne Fighter Radar	20	.001	“
4	Surface Fire Control Radar	10	.0001	“
5	Non - Cooperative Target Recognition	20	.001	“
6	Eastern Identification Friend or Foe	50	.01	“
7	IFF Interrogate	50	.001	“
8	IFF Reply	20	.001	“
9	Airborne Electronic Surveillance Measures (ESM)	500	.01	“
10	Surface ESM	300	.01	“
11	Infrared Detector	6000	.0001	“
12	Ownship Navigation	10	.00001	“
13	Data Link Tracking Errors	200	.01	“

Table A-1. Included Sensor Types and Associated Errors.

function [xnew] = timestep(F,x)

The TIMESTEP function receives the time increment matrix and returns the new state vector for a moving target. If the target is in SLM, the (F) matrix is used. If the target is turning, the (Fturn) matrix is sent. The TIMESTEP function simply multiplies the step matrix and the state vector.

[spxtrue,spmeas,sper,spR,in]=spitmeas(sensor,spx2,spman,ownfturn,simt,in)

The SPITMEAS function is used to iterate the sensor platform position (Sp) and to calculate the noisy measurement. First, a test is conducted to determine if the sensor platform is a maneuvering condition. If maneuvering, the (ownfturn) matrix is used to update the truth position of the platform. Assuming that the origin of the navigation solution is ownship cg, the previous value of Sp is used as the origin for the call to POLE2CART.

function [R,zret,derror] = meas(sensor,zre,Sp)

The MEAS function receives the x-y truth data as well as the sensor type and sensor position. A function call is made to the SENSERROR function to get the sensor errors for the current sensor. These errors are sent to the POLE2CART function for calculation of the noisy measurements. The function then stacks the noisy measurements in a vector (zret) and stacks the measurement covariance matrices (R) for return to the main program. This process is repeated for all targets and for each sensor type.

function [xhat33,count] = tracker(Q1,P1,R1,xhat1,z1,count,k,trkr)

A different tracker is called for every sensor that has been selected for the sensor platform. The TRACKER function is sent all the applicable covariance data, the previous state, the measurement, and the tracker type. To call the appropriate tracker, a switch-case statement is used with identical argument lists for all available trackers. The tracker types are listed with the SETUPTRACKER function.

function [xhat33,count] = pdaf(Q,P,R,xhat,zret,count)

function [xhat33,count] = immtrk(Q,P,R,xhat,zret,count)

function [xhat33,count] = kalman(Q,P,R,xhat,zret,count)

function [xhat33,count] = constGKF(Q,P,R,xhat,zret,count)

The PDAF function is the Probabilistic Data Association Filter, which can be used where clutter is present in the target tracking gate. The IMMTRK function contains the code for the Interacting Multiple Models tracker. All the needed variables are defined and initialized the first time the function is called. The variables that need to be saved from iteration to iteration are simply declared as global variables within the tracking function. This way, the variables are not destroyed each time the function returns the updated state. The MATLAB code is included in Appendix B. The function calls for the Kalman filter and the constant gain Kalman filter are identical.

function [linkdat,lstate] = linkreport(xhat,simt)

This function creates a simulated data link report for the targets and allows for addition of attribute data, such as Identification Friend or Foe data. The format of the report is very flexible and can be changed as desired by user to suit individual needs. [x,y,velocity,heading,time,iff1,iff2,iff3,iff4,iffc,emitter1,emitter2,ID,reportingunit]

Within the function, random noise is added to the to velocity and position to simulate navigation errors between platforms. Also, to simulate DLRP/gridlock errors, a fixed value of bias error is added to the position. The link report is updated every 5 delta-time cycles.

function simplot(mn,ns,xhat)

The SIMPLOT function allows plotting of the track data as the program is executing. If multiple Monte Carlo runs are used in execution of the simulation, this function is not recommended since execution is slowed dramatically.

function LSE = lsecalc(mn,state,posout,nsamples)

The function LSECALC calculates the Least Square Error between the state vectors and the truth positions. This data is stored for each target and each sensor at every time step.

function handl = errorplots(mn,ns,nsamples,delta,merror,LSmean)

The ERRORPLOTS function plots all the measurement error and Least Squares error for every target and sensor vs. time.

function hand = trackplot(mn,ns,posout,zoutmean,statemean)

The TRACKPLOT function plots the true position, the noisy measurements, and the track states for each track and each sensor. Also provided is a single plot for comparison of all sensors for each of the tracks.

Running the *Fusim* Simulation

To run fusim: Type *fusim* at the command line and use the default simulation time, sensor types, and targets. To change the simulation time, open the START.MAT file and change the first element to the desired simulation time (in minutes). The second element is the number of runs. The number of simulations should be set to 1 if using the SIMPLOT command in FUSIM. If numerous runs are desired for smoothing of the randomly generated error data, set the 2nd element to the desired number (recognizing that simulation time increases dramatically). To change the timestep, delta, set the third element to the desired delta in minutes.

To change the primary tracker from PDA to IMM or Kalman or Constant Gain Kalman, change the final element (1) to a (2), (3), or (4) respectively. The assignment of trackers to other sensors will fall in the order given in the SETUPTRACKERS function. To change the sensor types in the sensor platform, open the TARDAT.MAT file and change elements 10-13 to the desired sensor types as described in the SENSERROR function. Finally, any number of targets and target types can be entered in the same format as described previously.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. *FUSIM* PROGRAM MATLAB CODE

FUSIM PROGRAM MATLAB CODE

NAME	*Fusim*
INPUT	None
OUTPUT	None
PURPOSE	The *fusim* program is the main simulation file. This file initializes all variables, creates the simulation truth data, calls the measurement function, calls the trackers, calls the data link simulator, and calls the plotting functions.

```
clear all;
format long;
*******************************************************************
global F H Sp;
dumb = 0;
[A,A1,delta,t,nloops,nsamples,Spi,flag,ownsens] = start(dumb);
%Determine if ownship platform (the sensor platform) is maneuvering and set up for
%  separate tracking
spxi = [];
if flag == 1
        %  Ownship is maneuvering
        ownship = A(1,:);
        A(1,:) = [];
        [ownfturn,ownB] = fturn(ownship,delta);
        [spman,spxi] = manmatrix(ownship,ownB,t);
elseif flag == 0
        %  Ownship is not maneuvering
        ownship = A1(1,:);
        A1(1,:) = [];
        spxi = nonman(ownship);
end

[Fturn,B] = fturn(A,delta);
%  Fturn comes back with stacked Fturn matrices [Ft1;Ft2;Ft3;....]
% B comes back with all the other data stacked: [turng;hdg;speed;xi turng;hdg;speed;xi,...]
% The total number of targets will always be n + m, or mn
[n,b] = size(A1);
[m,b] = size(A);
mn = m+n;
nsens = [];
for k = 1:max(size(ownsens))
   if (ownsens(k) >= 1)
      nsens = [nsens,ownsens(k)];
   end
end
ns = max(size(nsens)); % This gives the total number of sensors on the sensor platform.
trkr = setuptrackers(ns); % Returns a vector with the desired trackers identified for %tracker.m
% Try getting an average for a number of runs:
for kk = 1:nloops
   Sp = Spi;    % Initializes the sensor position
   simt = 0.0; % Simt keeps track of the simulation time for the maneuvering targets
   posout = []; sppos = [];
   ztrue = [];
   zout = []; spzout = [];
   state = []; spstate = [];
   error = []; sperror = [];
   xin = [];
   x2n = []; spx2 = [];
   xim = [];
   x2m = [];
   xy = [];
   xhat1 = [];
   R1 = [];
   z1 = [];
   Q1 = [];
   P1 = [];
   derror1 = [];
   count = 1;
   linkout = [];
```

```
    LSerror = [];
%  FIRST TARGET TIMESTEP FOR INITIALIZATION:
%  Call the function 'nonman' to get the non-maneuvering target data in the same format
%  as the maneuvering target:  xi will be a matrix of [x;vx;y;vy;'s.......] side by side
%  mantime contains the column vectors of maneuver times for the maneuvering targets %only.

    sppos = [sppos,H*spxi];
    spx2 = F*spxi;
    sppos = [sppos,H*spx2];
    if n > 0
        xin = nonman(A1);
        x2n = zeros(size(xin));
        for k = 1:n
            x2n(:,k) = timestep(F,xin(:,k));
            ztrue = [ztrue;H*xin(:,k)];
        end
    end
    %  Get the xi's, which will be used for truth data [x;vx;y;vy], xim (maneuvering)
    if m > 0
        %  The function 'manmatrix' creates the matrix of maneuver times
        [mantime,xim] = manmatrix(A,B,t);
        x2m = zeros(size(xim));
        for k = 1:m
           ztrue = [ztrue;H*xim(:,k)];  % stacks the ztrues to make a 2*N x 1 vector
              x2m(:,k) = timestep(F,xim(:,k));
        end
        %  Update posout matrix with the stacked ztrues:
        %[x1;y1;x2;y2;}non-maneuvering x3;y3;x4;y4;...maneuvering]
        posout = [posout,ztrue];
     end
            %       INITIALIZATION:  Assume all targets are moving in a straight line!
            [rr,ll] = size(mantime);
            simt = simt + delta;
            %Sp = H*x2m(:,1);  %  ASSUMES THAT THE SENSOR PLATFORM IS THE FIRST
ELEMENT OF M-SET
            allx1 = [xin,xim];
```

```
allx2 = [x2n,x2m];
%  First, grab the sensor information to get the applicable errors:
%  Perform a Kalman filter initialization for every track - assume straight line!
ztemp = [];
for k = 1:ns
   ser = nsens(k);
   [Q,P,R,D,derror,xhat,zret] = init(allx1,allx2,delta,Spi,Sp,ser);
   xhat1(:,:,k) = xhat;
   R1(:,:,k) = R;
   z1(:,:,k) = zret;
   % For multiple sensors, the z's are stacked, not 2-D to begin with
   %  [(tgt1,sens1);(tgt2,sens1);.....(tgt1,sens2);(tgt2,sens2);...]
   Q1(:,:,k) = Q;
   P1(:,:,k) = P;
   derror1(:,:,k) = derror; %  Same for the errors, one number per point
   %  [(tgt1,sens1);(tgt2,sens1);.....(tgt1,sens2);(tgt2,sens2);...]
end
%  Call init for ownship:
[spP,spR,sper,Spst,spmeas] = spinit(spxi,spx2,Q,D);

% Create the noisy measurements matrix for output:
spzout = [spzout,spmeas];
zout = [zout,z1];  %  Noisy z measurements z1;z2......, and ns deep (1 page for each sensor)
z1 = [];
state = [state,xhat1];
spstate = [spstate,Spst];
Sptrue = H*spx2;  %  true x-y for ownship
Sp = H*Spst;      %  STATE predicted x-y for ownship
error = [error,derror1];
sperror = [sperror,sper];
derror1 = [];
%  Set X equal to the latest position.
xin = x2n;
xim = x2m;
ztrue = setztrue(m,n,xin,xim);
```

```
posout = [posout,ztrue];
% indx will be used to keep track of where the targets are in their maneuver cycle:
indx = ones(1,m);
in = 1;
  for ii = 3:nsamples

     % Iterate the targets:
     % NON-MANEUVERING TARGETS, A1, xin,
     if n > 0
        for k = 1:n
           xin(:,k) = timestep(F,xin(:,k));
        end
     end

     % MANEUVERING TARGETS, A, xim, xm
  if m > 0
     index = 1;
     for k = 1:m % must loop thru for each target before moving on
        % Check for zeros here?
        if indx(k) < rr % Executes every time
           LEG = mantime(indx(k),k);
           TURN = mantime(indx(k)+1,k);
        end
        if indx(k) == rr
           if mod(rr,2)==0
              LEG = mantime(indx(k),k);
              TURN = mantime(indx(k)+1,k);
           elseif mod(rr,2) == 1
              LEG = mantime(indx(k),k);
              TURN = LEG+1;
           end
        end
            if ( (simt <= LEG) | (simt > TURN) | (TURN == 0) )
               xim(:,k) = timestep(F,xim(:,k)); %  Use non-maneuvering F if tgt is SLM
            elseif (simt > LEG) & (simt <= TURN) | (LEG == 0) % What to do if turn is a zero? or
Leg?
```

```
            xim(:,k) = timestep(Fturn(index:(index+3),:),xim(:,k));
          end
        %  Reset LEG & TURN as needed for each target:
          if simt > TURN & TURN > 0 & indx(k) < rr
            indx(k) = indx(k) + 2;
          end

      index = index+4;
   end
end

  %********************      MEASUREMENT      **********************
   ztrue = setztrue(m,n,xin,xim);
   posout = [posout,ztrue];
   zre = [];
   zre = reshape(ztrue,2,mn);

   %  SENSOR PLATFORM MEASUREMENT AND TRACKING (NAV solution):
   sensor = 12;
   if flag == 0
      spxtrue = timestep(F,spx2);
      [spR,spmeas,sper] = meas(sensor,spxtrue,[0;0]);
   elseif flag == 1
      [spxtrue,spmeas,sper,spR,in] = spitmeas(sensor,spx2,spman,ownfturn,simt,in);
   end
   [Spst,count] = kalman1(Q,spP,spR,Spst,spmeas,count);  %  Calc sensor position state
   spx2 = spxtrue;
   sppos = [sppos,H*spxtrue];
   spstate = [spstate,Spst];
   sperror = [sperror,sper];
   spzout = [spzout,spmeas];
   Sp = H*Spst;  %  The x-y position of the sensor platform (nav solution)

   %  TARGET MEASUREMENT (for each sensor)
    for k = 1:ns
```

```
      sensor = nsens(k);  %  Sends the sensor type (numerical) to the measurement function
      zret = [];
      [R,zret,derror] = meas(sensor,zre,Sp);
      R1(:,:,k) = R;
      z1(:,:,k) = zret;
      derror1(:,:,k) = derror;
   end
error = [error,derror1];
% Create the noisy measurements matrix for output:
zout = [zout,z1];
%  zret are the measurement inputs for the tracker
%  Xhat is the matrix of updated states

%              *******************  TRACKER  **********************
for k = 1:ns
    [xhat33,count] = tracker(Q1(:,:,k),P1(:,:,k),R1(:,:,k),xhat1(:,:,k),z1(:,:,k),count,trkr(k));
    xhat1(:,:,k) = xhat33;
end
state = [state,xhat1];

%            **************    DATA LINK FUNCTION     *************************

%  The current sensor state for surveillance radar is used as a starting point, we're
%  getting farther away from the truth with link reports. Simulate by adding more error
%  Also, this function is only called every 5 loops (3 seconds right now).

if mod(ii,5) == 0
    [linkvector,lstate] = linkreport(xhat1(:,:,1),simt);
    %  lstate is simply the state vector for the data link report
    %  linkstate = [linkstate,lstate];
    %  lrep is in the form
[x;y;vel;IFF1;IFF2;IFF3;IFF4;IFFC;emitters;time;reportingunit,x2;.....]
    linkout = [linkout,linkvector];
end

count = count + 1;
```

```
      simt = simt + delta;
      %  SIMPLOT function for plotting on the fly
      %  could use this for plotting the fused result
      %simplot(mn,ns,xhat1,Spst);
    end
     %   ****                CALCULATE THE LEAST SQUARE ERROR:
     for k = 1:ns
        LSerr = lsecalc(mn,state(:,:,k),posout,nsamples);
        LSerror(:,:,k) = LSerr;
     end

     if kk == 1,
       merror = error;
       zoutmean = zout;
       statemean = state;
       spstatemean = spstate;
       sperrormean = sperror;
       spmeasmean = spzout;
       linkmean = linkout;
       LSmean = LSerror;
     else
       merror = merror + error;
       zoutmean = zoutmean + zout;
       statemean = statemean + state;
       spstatemean = spstatemean + spstate;
       sperrormean = sperrormean + sperror;
       spmeasmean = spmeasmean + spzout;
       linkmean = linkmean + linkout;
       LSmean = LSmean + LSerror;
     end

end

merror = merror/nloops;
zoutmean = zoutmean/nloops;
```

```
statemean = statemean/nloops;
spstatemean = spstatemean/nloops;
sperrormean = sperrormean/nloops;
spmeasmean = spmeasmean/nloops;
linkmean = linkmean/nloops;
LSmean = LSmean/nloops;
%
hand = trackplot(posout,zoutmean,statemean,sppos,spstatemean,linkmean);
handl = errorplots(delta,merror,LSmean,trkr);
```

NAME	*start*
INPUT	a dummy variable
OUTPUT	A,A1,delta, t,nloops,nsamples,Spi,flag,ownsens
PURPOSE	The *start* function reads the start.mat file to collect timing information about the simulation, initiates the matrices of targets, and initiates other needed matrices (F,H)

```
function [A,A1,delta,t,nloops,nsamples,Spi,flag,ownsens] = start(dumb)
%  OUTPUT VARIABLES:
%  A        Maneuvering target matrix
%  A1       Non-maneuvering target matrix
%  delta    delta time
%  t        total simulation time in seconds
%  nloops   number of simulation runs (for error analysis)
%  nsamples number of time steps in the simulation run
%  Spi      Initial position of the sensor platform
%  flag     Indicator of whether or not the sensor platform is maneuvering
%  ownsens  Vector of ownship sensor types
global H F;
%  START READS THE SIMULATION RUN PARAMETERS
  simstart = textread('start.mat');
%  simstart returns in the following fashion:  [t(simtime), nloops, deltatime, DLRP,
%       placeholder, tracker selection 1-4]
  t = simstart(1)*60; %  To put time into seconds for the simulation
  nloops = simstart(2);
%
%t = simtime(dumb);
```

```
%t = t*60;  %  to put time in seconds
%  determine the time increment and the total number of samples
uprate = simstart(3);
[delta,nsamples] = deltatime(t,uprate);
%  Start by getting file data on all targets:  A1 returns as a matrix of targets - N x 17
A1 = textread('tardat.mat');
A = [];
Spi = [A1(1,2);A1(1,4)];  %  First target of A1 is always ownship!! (the sensor platform)
ownsens = [A1(1,10) A1(1,11) A1(1,12) A1(1,13)];
%  Now, determine if the target is maneuvering or not by checking item 17
%   and creating a new matrix, A and deleting the maneuvering tgt rows from A1:
[d,e] = size(A1);
Atemp = [];
for i = 1:d
    if  A1(i,17) > 0
        A = [A;A1(i,:)];
        if i == 1
            flag = 1;
        end
    elseif A1(i,17) == 0
        Atemp = [Atemp;A1(i,:)];
    end
end
A1 = [];
A1 = Atemp;
%  Now A contains the targets that are maneuvering only, for use by the 'fturn' function
%  Stack the splat data on A.  Sensor platform is always the first row (or column in fturn)
%  If the sensor platform is not maneuvering, the splat info gets added to A1
H = [1,0,0,0;
   0,0,1,0];
F = [1, delta, 0, 0;
   0, 1, 0, 0;
   0, 0, 1, delta;
   0, 0, 0, 1];
```

NAME	*fturn*
INPUT	A, delta
OUTPUT	Fturn, B
PURPOSE	The *fturn* function reads the start.mat file to collect timing information about the simulation, initiates the matrices of targets, and initiates other needed matrices (F,H)

```
function [Fturn,B] = fturn(A,delta)
%  This function will create a matrix containing submatrices of Fturn's for each target
%  platform, including the sensor platform, if maneuvering.
B = [];
Btemp = [];
Fturn = [];
%  Initialize state vector of positions and velocities based on input values:
[nn,b] = size(A);
for jj = 1:nn
   if A(jj,16) == 0
      turng = 1.1;
   elseif A(jj,16) == 1
      turng = -1.1;
   end
   hdg = A(jj,5)*pi/180.0;
   speed = A(jj,6)*6076.0/3600.0;  % speed in feet/sec
   xi = [A(jj,2);speed*sin(hdg);A(jj,4);speed*cos(hdg)];
   Btemp = [turng;hdg;speed;xi];
   g = turng;
   %  Turn omega calculation
   w = g*32.174/speed;
   r = w*delta;
   s = sin(r);
   c = cos(r);
   Ft = [1,(s/w),0,((1-c)/w);
      0,c,0,-s;
      0,((1-c)/w),1,(s/w);
      0,s,0,c];
      Fturn = [Fturn;Ft];
   B = [B,Btemp];
```

```
end
```

NAME	*setuptrackers*
INPUT	ns
OUTPUT	trkr
PURPOSE	This function sets up the vector of desired trackers by reading the primary requested tracker and then assigning another tracker for the other sensors.

```
function trkr = setuptrackers(ns)
%  1 = PDAF
%  2 = IMM
%  3 = Kalman filter
%  4 = Const Gain Kalman
%  5 = Kalman filter #2 (for navigation)
trkr = [];
simstart = textread('start.mat');
primarytracker = simstart(6);  %  Reads the desired tracker for the primary radar tracking
for k = 1:ns
    trkr = [trkr,primarytracker];
    if primarytracker == 1
        temp = 2;
    elseif primarytracker == 2
        temp = 3;
    elseif primarytracker == 3
        temp = 4;
    elseif primarytracker == 4
        temp = 1;
    end
    primarytracker = temp;
end
```

NAME	*manmatrix*
INPUT	A,B,t
OUTPUT	mantime,xim
PURPOSE	This function sets up the Maneuver Time Matrix based on the size of the largest maneuver vector and calls the maneuvertime function with the needed info. Based on the target vector and simulation time, determine the times for maneuvers and creates a vector to hold the cumulative times. For consistency, xim is returned, similar to the nonman function

```
function [mantime,xim] = manmatrix(A,B,t)
 time = t;
[N,b] = size(A);
mantime = [];
mt = [];
for kk = 1:N
   legt = round(A(kk,15)*60);
   turnt = round(A(kk,17)*60);
   mt = maneuvertime(time,legt,turnt);
   if kk == 1
      r = length(mt);
      longestr = r;
      mantime = zeros(r,N);
      mantime(:,1) = mt;
   end
   if kk > 1
      s = length(mt);
      r = longestr;
      if s > r
         mantime = [mantime;zeros((s-r),N)];  %  Adds s-r rows of zeros to mantime
         mantime(:,kk) = mt;
         longestr = s;
      elseif s < r
         mantime(:,kk) = [mt;zeros((r-s),1)];
      elseif s == r
         mantime(:,kk) = mt;
      end
```

```
    end

end
%  Peel off the x's
xim = B(4:7,:);
```

NAME	*maneuvertime*
INPUT	t,legt,turnt
OUTPUT	Mt
PURPOSE	This function determines if the target is maneuvering based on the times given and uses legt and turnt to calculate the time at which the target turns and stops turn: t enters the function in seconds, returns mt which is a vector of turn times for a maneuvering target, or else it returns an empty vector for a target that will not maneuver.

```
function [mt] = maneuvertime(t,legt,turnt)

mt = [];

if turnt == 0
    disp('Are you nuts?  This target is not maneuvering, Program Terminated!');
    return
end
totaltime = 0;
step = 0;
k = 1;
while (totaltime < t)
    if (mod(k,2) == 1)
        step = legt;
    elseif (mod(k,2) == 0)
        step = turnt;
    end
    totaltime = totaltime + step;
    if totaltime > t
        totaltime = t;
```

```
   end
   mt = [mt;totaltime];
   k = k + 1;
end
[m,o] = size(mt);
   if (mod(m,2) == 1)  % if number of elements is odd
      mt = [mt;0];
   end

%  Returns the vector with the times for maneuver in the following format:
%       [time at which turn1 executes;
%          time to return to straight line
%          time to start next turn
%          .......]
```

NAME	*nonman*
INPUT	A1
OUTPUT	xin
PURPOSE	This function will extract the position and velocity data form the A1 matrix of non-maneuvering targets. Of course, this also includes fixed targets. Initializes state vector of positions and velocities based on input values:

```
function [xin] = nonman(A1)
[N,b] = size(A1);
xin = zeros(size(4,N));
for jj = 1:N
   hdg = A1(jj,5)*pi/180.0;
   if A1(jj,6) > 0
      speed = A1(jj,6)*6076.0/3600.0;  % speed in feet/sec
   elseif A1(jj,6) == 0
      speed = 0;
   end
   xin(1:4,jj) = [A1(jj,2);speed*sin(hdg);A1(jj,4);speed*cos(hdg)];
end
```

NAME	*ownxi*
INPUT	ownship
OUTPUT	Spxi
PURPOSE	This function extracts the position and velocity data from the ownship vector, initializes state vector of positions and velocities based on input values

```
function [spxi] = ownxi(ownship)
   hdg = ownship(5)*pi/180.0;
   speed = ownship(6)*6076.0/3600.0;  % speed in feet/sec
   spxi = [ownship(2);speed*sin(hdg);ownship(4);speed*cos(hdg)];
end
```

NAME	*fturn*
INPUT	allx1,allx2,delta,Spi,Sp,sensor
OUTPUT	Q,P,R,D,derror,xhat,zret
PURPOSE	This function provides the initial gain for use by any tracking filter, assumes straight line motion, and provides noisy measurements via pole2cart.

```
function [Q,P,R,D,derror,xhat,zret] = init(allx1,allx2,delta,Spi,Sp,sensor)
% Input variables:
%   H       extraction matrix
%   F       Straight line motion matrix
%   allx1      initial x positions
%   allx2      incremented x positions
%   delta   time delta
%   Sensor  (vector) with applicable sensor for senserror
%
% Output variables:
%   Q       q matrix
%   P       initial covariance
%   derror  error matrix
%   xhat    initial x hat matrix [x;vx;y;vy,x;vx;y;vy,.....]  one xhat for each target.
%   zret    xy measurement vector (noisy)
global H F;
% INITIALIZATION STEPS:
```

```
P = [];  %  Size of P will be 4*N x 4 (a 4x4 for each target)
y = [];
y2 = [];
xhat = [];
a = delta^2/2;
b = delta^3/3;
% time for some special purpose matrices:
   Q = [b, a, 0, 0;
      a, delta, 0, 0;
      0, 0, b, a;
      0, 0, a, delta];
   D = [1, 0, 0, 0;
      1/delta, 0, -1/delta, 0;
      0, 1, 0, 0;
      0 1/delta, 0, -1/delta];

zret = [];
ztemp = [];
derror = [];
derror2 = [];
R = [];
Rinit = [];
sigr = 0.0;
sigb = 0.0;
Sv = [];
[sigr,sigb,Sv] = senserror(sensor);
[aa,bb] = size(allx1);
%  Measurement 1:  Add noise to initial positions:
    for h = 1:bb
        %  Take a noisy measurement for all the targets (man and non-man)!
        [zcart,R1,disterror] = pole2cart(H*allx1(:,h),Spi,sigr,sigb,Sv);
        zret = [zret;zcart];
        y = [y,zcart];
        derror = [derror;disterror];
        Rinit = [Rinit;R1];
    end
%  Measurement 2:
    for h = 1:bb
        %  Take a noisy measurement!
        [zcart,R2,disterror] = pole2cart(H*allx2(:,h),Sp,sigr,sigb,Sv);
        ztemp = [ztemp;zcart];
        y2 = [y2,zcart];
        R = [R;R2];
        derror2 = [derror2;disterror];
    end

zret = [zret,ztemp];
derror = [derror,derror2];
y = [y;y2];
ytemp = zeros(4,1);

for h = 1:bb
      %  vector yflipped contains the z's stacked z(2)/z(1)
      ytemp = y(:,h);
      yflipped = [ytemp(3);ytemp(4);ytemp(1);ytemp(2)];
      plantnoise = Rinit(2*h-1:2*h,:)+H*inv(F)*Q*(inv(F))'*H';
      cov = [R(2*h-1:2*h,:) zeros(2,2);
         zeros(2,2) plantnoise];
      P1 = D*cov*D';
      P = [P;P1];
      xhat1 = D*yflipped;
      xhat = [xhat;xhat1];  %  xhat is returned as a stacked vector of [x;vx;y;vy.......]
end
```

NAME	*pole2cart*
INPUT	ztrue,sp,sr,sb,Sv
OUTPUT	z,R,disterror
PURPOSE	This function processes the measurements from the sensors by converting to polar coords and adding the supplied error, then converting back to Cartesian.

```
      function [z,R,disterror] = pole2cart(ztrue,sp,sr,sb,Sv)
%  Input Arguments
%  ztrue:          True target position
%  sp:    True sensor position
%  sr:             Range Standard deviation error
%  sb:             Bearing standard deviation error
%  sv:             sensor variance matrix (2x2)

%  Outputs:
%  z:                          Output measurement with error in cart coords
%  R:                          Covariance Matrix of the meas in cart coords
%  disterror: Distance error between meas and true posit

%  Convert current position to polar coordinates and add sensor noise

%  Generating a noisy measurement from the sensor:
z = ztrue-sp;                   %  position relative to the sensor
r = sqrt(z(1)^2 + z(2)^2);  % Range from sensor
b = atan2(z(2),z(1));                   % Bearing from sensor
r = r+sr*randn;                                  % Range + sensor noise
b = b+sb*randn;                                  % Bearing + sensor noise

%  Convert the measurement to Cartesian coordinates
z = [r*cos(b);r*sin(b)]+sp;
%  Now make the measurement covariance in Cartesian coords:
fx = [cos(b) -r*sin(b);sin(b) r*cos(b)];
R = fx*Sv*fx';
%  Compute the measurement error in Cartesian coordinates
ztilde = ztrue - z;
disterror = sqrt(ztilde'*ztilde);
```

NAME	*spinit*
INPUT	spxi,spx2,Q,D
OUTPUT	spP,spR,sperror,spstate,spmeas
PURPOSE	This function provides the initial gain and state estimation for ownship navigation, assumes straight line motion, and provides noisy measurements via pole2cart.

```
function [spP,spR,sperror,spstate,spmeas] = spinit(spxi,spx2,Q,D)
%  Input variables:
%   spx1      initial x positions
%   spx2      incremented x positions
%   Q         Q matrix
%   D         D matrix
%  Output variables:
%   spP        initial covariance
%   spR        sensor error covariance
%   sperror    error matrix
%   spstate    initial xhat matrix [x;vx;y;vy,x;vx;y;vy,.....]  one xhat for each target.
%   spmeas     noisy platform measurements
global H F;
%  INITIALIZATION STEPS:
spP = [];
spR = [];
y = [];
y2 = [];
spstate = [];
spQ = 1000*Q;
sensorposition = [0;0]; % Assumes that the origin of the nav solution is ownship cg
spmeas = [];
sperror = [];
sensor = 12;
[sigr,sigb,Sv] = senserror(sensor);
%sigr = 20.0;  %  Feet
%sigb = 0.001; %  Radians from ownship
%Sv = diag([sigr^2;sigb^2]);
%  Measurement 1:  Add noise to initial position:
    %  Take a noisy measurement
```

```
        [zcart1,spR1,disterror] = pole2cart(H*spxi,sensorposition,sigr,sigb,Sv);
        spmeas = [spmeas,zcart1];
        sperror = [sperror,disterror];

%  Measurement 2:
        %  Take a noisy measurement
        [zcart2,spR2,disterror] = pole2cart(H*spx2,sensorposition,sigr,sigb,Sv);
        spmeas = [spmeas,zcart2];
        sperror = [sperror,disterror];
%  vector yflipped contains the z's stacked z(2)/z(1)
      yflipped = [zcart2(1);zcart2(2);zcart1(1);zcart1(2)];
      plantnoise = spR1 + H*inv(F)*spQ*(inv(F))'*H';
      cov = [spR2 zeros(2,2);
         zeros(2,2) plantnoise];
      spP = D*cov*D';
      spstate = D*yflipped;
```

NAME	*setztrue*
INPUT	m,n,xin,xim
OUTPUT	ztrue
PURPOSE	This function simply extracts the true x,y data from the maneuvering/non-maneuvering vectors.

```
function ztrue = setztrue(m,n,xin,xim)
global H;
ztrue = [];
%  NON-MANEUVERING
if n > 0
    for k = 1:n
        ztrue = [ztrue;H*xin(:,k)];
    end
end
%  MANEUVERING
if m > 0
   for k = 1:m
       ztrue = [ztrue;H*xim(:,k)];
   end
end
```

NAME	*senserror*
INPUT	sensor
OUTPUT	sigr,sigb,Sv
PURPOSE	This function matches the type of sensor to the corresponding errors. For each of the sensors on the sensor platform, the range and bearing errors are returned along with the sensor variance matrix.

```
function [sigr,sigb,Sv] = senserror(sensor)

% NONE OF THE FORMAT TYPES ARE CORRECTED!!  ALL SENSORS ARE IN X-Y
COORDS

% Input variables:

%   sensor  # representing the sensor type

%   1   Airborne search radar

%   2   Shipboard search radar

%   ...

%   See the switch-case statement for the rest of the descriptions.


% Output variables:

%   sigr    sigma range

%   sigb    sigma bearing

%   Sv      sensor variance matrix

switch sensor

     case 1,

        % AIRBORNE SURVEILLANCE RADAR 2-D

        sigr = 100;

        sigb = .005;

        Sv = diag([sigr^2;sigb^2]);

     case 2,

        % SURFACE SURVEILLANCE RADAR 2-D

        sigr = 20;

        sigb = .0001;

        Sv = diag([sigr^2;sigb^2]);

     case 3,
        %  AIRBORNE FIGHTER RADAR
        sigr = 20;
        sigb = .001;
        Sv = diag([sigr^2;sigb^2]);
     case 4,
        %  SURFACE FIRE CONTROL RADAR
        sigr = 10;
        sigb = .00001;
        Sv = diag([sigr^2;sigb^2]);
     case 5,
        %  NCTR
        sigr = 20;
        sigb = .0001;
        Sv = diag([sigr^2;sigb^2]);
     case 6,
        %  EIFF
        sigr = 20;
        sigb = .0001;
        Sv = diag([sigr^2;sigb^2]);
     case 7,
        %  IFF INTERROGATE
        sigr = 50;
        sigb = .001;
        Sv = diag([sigr^2;sigb^2]);
     case 8,
        %  IFF REPLY ONLY (NOT USED YET)
        sigr = 20;
        sigb = .0001;
        Sv = diag([sigr^2;sigb^2]);
     case 9,
        %  AIRBORNE ESM
        sigr = 500;
        sigb = .01;
        Sv = diag([sigr^2;sigb^2]);
     case 10,
        % SURFACE ESM
        sigr = 300;
        sigb = .01;
        Sv = diag([sigr^2;sigb^2]);
    case 11,
        % IR DETECTOR
        sigr = 6000;
        sigb = .0001;
        Sv = diag([sigr^2;sigb^2]);
    case 12,
        % OWNSHIP NAVIGATION!!
        sigr = 10;
        sigb = 0.0001;
        Sv = diag([sigr^2;sigb^2]);
    case 13,
        % DATA LINK REPORT TRACKING ERRORS
        sigr = 200;
        sigb = .01;
        Sv = diag([sigr^2;sigb^2]);
    otherwise,
        disp('Unknown sensor')
    end
```

NAME	*timestep*
INPUT	F,x
OUTPUT	xnew
PURPOSE	The timestep function receives the time increment (F-matrix) and returns the new state vector for a moving target or platform, can be F or Fturn.

```
function [xnew] = timestep(F,x)
    xnew = F*x;
```

NAME	*spitmeas*
INPUT	sensor,spx2,spman,ownfturn,simt,in
OUTPUT	spxtrue,spmeas,sper,spR,in
PURPOSE	This function will iterate the sensor platform and calculate the noisy measurement

```
function [spxtrue,spmeas,sper,spR,in] = spitmeas(sensor,spx2,spman,ownfturn,simt,in);
global H F sLEG sTURN;
origin = [0;0];   % Assumes that the origin of the nav solution is ownship cg
spmeas = [];
sper = [];
spR = [];
sLEG = 0;
sTURN = 0;
%  spman has the maneuver times, only need to do one test:
[rr,ll] = size(spman);  % so rr = the number of maneuver times
%  Iterate the truth position:
      %  Check for zeros here?
      if in < rr  %  Executes every time
        sLEG = spman(in);
        sTURN = spman(in+1);
      end
      if in == rr
        if mod(rr,2)==0
          sLEG = spman(in);
          sTURN = spman(in+1);
        elseif mod(rr,2) == 1
          sLEG = spman(in);
          sTURN = sLEG+1;
        end
      end
         if ( (simt <= sLEG) | (simt > sTURN) | (sTURN == 0) )
           spxtrue = timestep(F,spx2);  %   Use non-maneuvering F if tgt is SLM
         elseif (simt > sLEG) & (simt <= sTURN) | (sLEG == 0) %  What to do if turn is a zero?
or Leg?
           spxtrue = timestep(ownfturn,spx2);
         end
```

```
          %  Reset LEG & TURN as needed for each target:
            if simt > sTURN & sTURN > 0 & in < rr
               in = in + 2;
            end

%  Measurement:  Add noise to position:
 [sigr,sigb,Sv] = senserror(sensor);
[spmeas,spR,sper] = pole2cart(H*spx2,origin,sigr,sigb,Sv);
```

NAME	*meas*
INPUT	sensor,zre,Sp
OUTPUT	R,zret,derror
PURPOSE	This function takes the noisy measurements and returns the matrix of noisy measurements (depending on the sensor type)

```
function [R,zret,derror] = meas(sensor,zre,Sp)
%  INPUTS:
%   sensor: sensor type (just search radar for now)
%   zre:      x matrix of true x,y measurements
%   Sp:     sensor platform position
%  OUTPUTS:
%  xret     output matrix of noisy x,y coordinates
%  disterror    actual distance errors for each measurement
%  R        error covariance for the measurement of each target [R1;R2;R3;R4;R5....]
%global MEAS_R zpol;
zret = [];
derror = [];
sigr = 0.0;
sigb = 0.0;
Sv = [];
[sigr,sigb,Sv] = senserror(sensor);
[aa,bb] = size(zre);
R = zeros(2*bb,2);
for h = 1:bb
```

```
    % Take a noisy measurement!
    [zcart,R1,disterror] = pole2cart(zre(:,h),Sp,sigr,sigb,Sv);
    zret = [zret;zcart];
    derror = [derror;disterror];
    R(2*h-1:2*h,:) = R1;
end
```

NAME	*tracker*
INPUT	Q1,P1,R1,xhat1,z1,count,trkr
OUTPUT	xhat33,count
PURPOSE	Right now, this function only calls the appropriate tracker with all the applicable track data.

```
function [xhat33,count] = tracker(Q1,P1,R1,xhat1,z1,count,trkr)
    switch trkr
        case 1,
            [xhat33,count] = pdaf(Q1,P1,R1,xhat1,z1,count);

        case 2,
            [xhat33,count] = immtrk(Q1,P1,R1,xhat1,z1,count);
        case 3,
            [xhat33,count] = kalman(Q1,P1,R1,xhat1,z1,count);
        case 4,
            [xhat33,count] = constGKF(Q1,P1,R1,xhat1,z1,count);
        case 5,
            [xhat33,count] = kalman1(Q1,P1,R1,xhat1,z1,count);
        otherwise,
            disp('Undefined Tracker')
        end
```

NAME	*pdaf*
INPUT	Q,P,R,xhat,zret,count
OUTPUT	xhat33,count
PURPOSE	Tracker for *fusim*, will perform the following functions: PREDICT, based on previous stuff (stored as globals) UPDATE, based on current measurement

```
% Thesis Tracker: Probabilistic Data Association Filter
function [xhat33,count] = pdaf(Q,P,R,xhat,zret,count)
global H F M I Qp Pd Pg Pp lambda sig2r sig2b gatev;
global Sp;
if count > 1
  xhatone = zeros(size(xhat));
  m = 2;
  for k = 1:M

    a = 2*k-1;
    b = 2*k;
    c = 4*k-3;
    d = 4*k;
    xhatest = [];
    Pproj = [];
    % Now for the weighted predictions:
    xhatone(c:d,1) = F*xhat(c:d,1);
    Pproj = F*Pp(c:d,:)*F' + Qp(c:d,:); %okay so far

    % Calculate the b using non-parametric assumptions (diffuse prior model)
    % **************************      ************************
              e = [];
              nu = [];
              nuk = zeros(2,1);
              beta = [];
              mess = zeros(2,2);

    % Semi-major and semi-minor axis in feet:
            %minor = 600;
            %major = 1500;
            %Vk = 2*pi*minor*major;
            %lambda = m/Vk;
    PreS = H*Pproj*H';
    %  Calculate the innovation stuff for the actual measurement, based on meas S:
    Sk = PreS + R(a:b,:);
    Sinv = inv(Sk);
    K = Pproj*H'*Sinv;  %  W!!
    W = K;
    bb = lambda*((det(2*pi*Sk))^.5)*(1-Pd*Pg)/Pd;

    sum = bb;
    knu = zret(a:b,1) - H*xhatone(c:d,1);
    %knu = zcart - zproj;
            e(1) = exp(-0.5*knu'*Sinv*knu);
            nu(:,1) = knu;
            sum = sum+e(1);

    if m > 1,
       for j = 2:m
          [zclutter,Rclut,er] = pole2cart(zret(a:b,1),Sp,sig2r,sig2b,gatev);
          zset = zclutter;
               knu = zset - H*xhatone(c:d,1);
                        e(j) = exp(-0.5*knu'*Sinv*knu);
                        nu(:,j) = knu;
                        sum = sum+e(j);
               end
            end

            for j = 1:m
       v = nu(:,j);
       beta(j) = e(j)/sum;
       nuk = nuk + beta(j)*v;
    end
```

```
    for j = 1:m
       v = nu(:,j);
       mess = mess + (beta(j)*v*v');
    end
    mess = mess-nuk*nuk';

    betazero = bb/sum;
            Ptilde = W*mess*W';
    %  ********************************************************

    Pc = Pproj-W*Sk*W';
    Pp(c:d,:) = betazero*Pproj + (1-betazero)*Pc + Ptilde;
    xhatest = xhatone(c:d,1) + W*nuk;
    xhat33(c:d,1) = xhatest;
  end
  return
elseif count == 1

 Pd = 1.0;
 Pg = .99;
 m = 2;
%  Used only for generating "noisier" points:
 sig2r = 900;
 sig2b = 2.0*pi/180;
 gatev = [sig2r^2,0;
   0,sig2b^2];

 qsquared = 1000;
 I = eye(4);
 Pp = zeros(size(P));
 Pp(:,:) = P;
 M = max(size(xhat));
 M = M/4;
 Qp = zeros(size(P));
```

```
xhatone = zeros(size(xhat));

for k = 1:M

   a = 2*k-1;
   b = 2*k;
   c = 4*k-3;
   d = 4*k;
   Qp(c:d,:) = qsquared*Q;
   xhatest = [];
   Pproj = [];
   %  Now for the weighted predictions:
   xhatone(c:d,1) = F*xhat(c:d,1);
   Pproj = F*Pp(c:d,:)*F' + Qp(c:d,:);  %okay so far

   %  Calculate the b using non-parametric assumptions (diffuse prior model)
   %  ***************************       *************************
                 e = [];
                 nu = [];
                 nuk = zeros(2,1);
                 beta = [];
                 mess = zeros(2,2);

   %  Semi-major and semi-minor axis in feet:
          minor = 600;
          major = 1500;
          Vk = 2*pi*minor*major;
          lambda = m/Vk;
   PreS = H*Pproj*H';
   %  Calculate the innovation stuff for the actual measurement, based on meas S:
   Sk = PreS + R(a:b,:);
   Sinv = inv(Sk);
   K = Pproj*H'*Sinv;  %  W!!
   W = K;
   bb = lambda*((det(2*pi*Sk))^.5)*(1-Pd*Pg)/Pd;
```

```
sum = bb;
knu = zret(a:b,1) - H*xhatone(c:d,1);
e(1) = exp(-0.5*knu'*Sinv*knu);
          nu(:,1) = knu;
          sum = sum+e(1);

if m > 1,
    for j = 2:m
        [zclutter,Rclut,er] = pole2cart(zret(a:b,1),Sp,sig2r,sig2b,gatev);
        zset = zclutter;
              knu = zset - H*xhatone(c:d,1);
                          e(j) = exp(-0.5*knu'*Sinv*knu);
                          nu(:,j) = knu;
                          sum = sum+e(j);
              end
          end

          for j = 1:m
    v = nu(:,j);
    beta(j) = e(j)/sum;
    nuk = nuk + beta(j)*v;
end
for j = 1:m
    v = nu(:,j);
    mess = mess + (beta(j)*v*v');
end
mess = mess-nuk*nuk';

betazero = bb/sum;
          Ptilde = W*mess*W';
%  ***********************************************************

Pc = Pproj-W*Sk*W';
Pp(c:d,:) = betazero*Pproj + (1-betazero)*Pc + Ptilde;
```

```
        xhatest = xhatone(c:d,1) + W*nuk;
        xhat33(c:d,1) = xhatest;

      end
      return
  end
```

NAME	*immtrk*
INPUT	Q,P,R,xhat,zret,count
OUTPUT	xhat33,count
PURPOSE	Tracker for *fusim*, will perform the following functions: PREDICT, based on previous stuff (stored as globals) UPDATE, based on current measurement

```
% Thesis Tracker:  Interacting Multiple Models (2) Filter
function [xhat33,count] = immtrk(Q,P,R,xhat,zret,count)
global H F M I Qi Qturn Pi Pturn xhatone xhattwo alpha beta row musave;

if count > 1

    for k = 1:M
                    % PREDICTION STEP:
      a = 2*k-1;
      b = 2*k;
      c = 4*k-3;
      d = 4*k;
      mu = musave(:,k);
      % Pre-process the cbars and mu's:
      cbarone = row(1,1)*mu(1)+row(2,1)*mu(2);
      cbartwo = row(1,2)*mu(1)+row(2,2)*mu(2);
      % Set up the calculations for xhat-zero-one
      xhatzone = xhatone(c:d,1)*(row(1,1)*mu(1)/cbarone) +
xhattwo(c:d,1)*(row(2,1)*mu(2)/cbarone);
      xhatztwo = xhatone(c:d,1)*(row(1,2)*mu(1)/cbartwo) +
xhattwo(c:d,1)*(row(2,2)*mu(2)/cbartwo);

      muoneone = row(1,1)*mu(1)/cbarone;
```

```
                mutwoone = row(2,1)*mu(2)/cbarone;
                muonetwo = row(1,2)*mu(1)/cbartwo;
                mutwotwo = row(2,2)*mu(2)/cbartwo;

                xtilde11 = xhatone(c:d,1)-xhatzone;
                xtilde21 = xhattwo(c:d,1)-xhatzone;
                xtilde12 = xhatone(c:d,1)-xhatztwo;
                xtilde22 = xhattwo(c:d,1)-xhatztwo;

                Pzeroone = muoneone*(Pi(c:d,:)+xtilde11*xtilde11') +
mutwoone*(Pturn(c:d,:)+xtilde21*xtilde21');
                Pzerotwo = muonetwo*(Pi(c:d,:)+xtilde12*xtilde12') +
mutwotwo*(Pturn(c:d,:)+xtilde22*xtilde22');

                %  Now for the weighted predictions:
                xhatone(c:d,1) = F*xhatzone;
                xhattwo(c:d,1) = F*xhatztwo;
                Pone = F*Pzeroone*F' + Qi(c:d,:);
                Ptwo = F*Pzerotwo*F' + Qturn(c:d,:);

                %           Kalman Straight line:
                K = Pone*H'*inv(H*Pone*H'+R(a:b,:));
                %  Kalman Turn:
                Kturn = Ptwo*H'*inv(H*Ptwo*H'+R(a:b,:));
                Pi(c:d,:) = (I-K*H)*Pone*(I-K*H)'+K*R(a:b,:)*K';
                Pturn(c:d,:) = (I-Kturn*H)*Ptwo*(I-Kturn*H)'+Kturn*R(a:b,:)*Kturn';
                S1 = H*Pone*H'+ R(a:b,:);
                S2 = H*Ptwo*H'+ R(a:b,:);

                %  MEASUREMENT UPDATE
                %  Calculate the state estimate for straight line and turning:
                %  Straight line
                zt1 = zret(a:b,1) - H*xhatone(c:d,1);
                xhatest = xhatone(c:d,1)+K*(zt1);
                %  Turning
                zt2 = zret(a:b,1) - H*xhattwo(c:d,1);
```

```
            xhatturn = xhattwo(c:d,1)+Kturn*(zt2);

            %  Scores:
            %m = 2.0;
            Lambda1 = exp(-((zt1'*inv(S1)*zt1)/2.0))/( 2*pi*norm(S1)^.5 );
            Lambda2 = exp(-((zt2'*inv(S2)*zt2)/2.0))/( 2*pi*norm(S2)^.5 );
            see = Lambda1*cbarone + Lambda2*cbartwo;
            %Store for next iteration
            xhatone(c:d,1) = xhatest;
            xhattwo(c:d,1) = xhatturn;
            %  Combine the data for output
            xhat33(c:d,1) = mu(1)*xhatest + mu(2)*xhatturn;

            mustr = Lambda1*cbarone/see;
            muturn = Lambda2*cbartwo/see;
            musave(:,k) = [mustr;muturn];
        end
        %musave
elseif count == 1

    %  Set the probability of a turn if in SLM (alpha) and the prob of stop turn (beta)
    %  if in a turn:
    alpha = 0.1;
    beta = 0.33333;
    qsquared = 1;
    turnq2 = 100000;
    row = [(1-alpha),alpha;beta,(1-beta)];
    xhatone = xhat;
    xhattwo = xhatone;
    I = eye(4);
    Pi = zeros(size(P));
    Pi(:,:) = P;
    Pturn = zeros(size(P));
    Pturn(:,:) = P;
    M = max(size(xhat));
```

```
M = M/4;
% Initial State Likelihood: (straight line track)
musave = zeros(size(2,M));
mu = [1;0];
Qi = zeros(size(P));
Qturn = zeros(size(P));
for k = 1:M
    musave(1:2,k) = mu;
end

for k = 1:M
                    % INITIAL PREDICTION STEP ONLY!!:

    a = 2*k-1;
    b = 2*k;
    c = 4*k-3;
    d = 4*k;
    Qi(c:d,:) = qsquared*Q;
    Qturn(c:d,:) = turnq2*Q;
    % Pre-process the cbars and mu's:

    cbarone = row(1,1)*mu(1)+row(2,1)*mu(2);
    cbartwo = row(1,2)*mu(1)+row(2,2)*mu(2);
    % Set up the calculations for xhat-zero-one
    xhatzone = xhatone(c:d,1)*(row(1,1)*mu(1)/cbarone) +
xhattwo(c:d,1)*(row(2,1)*mu(2)/cbarone);
    xhatztwo = xhatone(c:d,1)*(row(1,2)*mu(1)/cbartwo) +
xhattwo(c:d,1)*(row(2,2)*mu(2)/cbartwo);

    muoneone = row(1,1)*mu(1)/cbarone;
    mutwoone = row(2,1)*mu(2)/cbarone;
    muonetwo = row(1,2)*mu(1)/cbartwo;
    mutwotwo = row(2,2)*mu(2)/cbartwo;

    xtilde11 = xhatone(c:d,1)-xhatzone;
    xtilde21 = xhattwo(c:d,1)-xhatzone;
```

```
                    xtilde12 = xhatone(c:d,1)-xhatztwo;
                    xtilde22 = xhattwo(c:d,1)-xhatztwo;


                    Pzeroone = muoneone*(Pi(c:d,:)+xtilde11*xtilde11') +
mutwoone*(Pturn(c:d,:)+xtilde21*xtilde21');
                    Pzerotwo = muonetwo*(Pi(c:d,:)+xtilde12*xtilde12') +
mutwotwo*(Pturn(c:d,:)+xtilde22*xtilde22');


                    %  Now for the weighted predictions:
                    xhatone(c:d,1) = F*xhatzone;
                    xhattwo(c:d,1) = F*xhatztwo;
                    Pone = F*Pzeroone*F' + Qi(c:d,:);
                    Ptwo = F*Pzerotwo*F' + Qturn(c:d,:);


                    %   Kalman Straight line:
                    K = Pone*H'*inv(H*Pone*H'+R(a:b,:));
                    %  Kalman Turn:
                    Kturn = Ptwo*H'*inv(H*Ptwo*H'+R(a:b,:));
                    Pi(c:d,:) = (I-K*H)*Pone*(I-K*H)'+K*R(a:b,:)*K';
                    Pturn(c:d,:) = (I-Kturn*H)*Ptwo*(I-Kturn*H)'+Kturn*R(a:b,:)*Kturn';
                    S1 = H*Pone*H'+ R(a:b,:);
                    S2 = H*Ptwo*H'+ R(a:b,:);


                    %  MEASUREMENT UPDATE
                    %  Calculate the state estimate for straight line and turning:
                    %  Straight line
                    zt1 = zret(a:b,1) - H*xhatone(c:d,1);
                    xhatest = xhatone(c:d,1)+K*(zt1);
                    %  Turning
                    zt2 = zret(a:b,1) - H*xhattwo(c:d,1);
                    xhatturn = xhattwo(c:d,1)+Kturn*(zt2);


                    xhat33(c:d,1) = mu(1)*xhatest + mu(2)*xhatturn;


                    %  Scores:
                    %m = 2.0;
```

```
          Lambda1 = exp(-((zt1'*inv(S1)*zt1)/2.0))/( 2*pi*norm(S1)^.5 ); %(2*pi)^(m/2)*norm(S1)^.5
          Lambda2 = exp(-((zt2'*inv(S2)*zt2)/2.0))/( 2*pi*norm(S2)^.5 ); %(2*pi)^(m/2)*norm(S2)^.5
          see = Lambda1*cbarone + Lambda2*cbartwo;
          mustr = Lambda1*cbarone/see;
          muturn = Lambda2*cbartwo/see;
          musave(:,k) = [mustr;muturn];

          xhatone(c:d,1) = xhatest;
          xhattwo(c:d,1) = xhatturn;

     end
     return
end
```

NAME	*kalman*
INPUT	Q,P,R,xhat,zret,count
OUTPUT	xhat33,count
PURPOSE	Tracker for *fusim*, will perform the following functions: PREDICT, based on previous stuff (stored as globals) UPDATE, based on current measurement

```
%  Thesis Tracker:  Kalman Filter
function [xhat33,count] = kalman(Q,P,R,xhat,zret,count)
global H F Mk I Pk Qk;

if count > 1
     xhat32 = zeros(size(xhat));
     for k = 1:Mk
                    % PREDICTION STEP:
       Ptemp = [];
       a = 2*k-1;
       b = 2*k;
       c = 4*k-3;
       d = 4*k;
       Ptemp = (F*Pk(c:d,:)*F' + Qk(c:d,:));
```

```
            Pk(c:d,:) = Ptemp;
            xhat32(c:d,1) = F*xhat(c:d,1);

                    %  UPDATE STEP:
            K = Pk(c:d,:)*H'*inv(H*Pk(c:d,:)*H' + R(a:b,:));  %  S1v = R
            Ppre(c:d,:) = (I-K*H)*Pk(c:d,:)*(I-K*H)'+K*R(a:b,:)*K';
            xhat33(c:d,1) = xhat32(c:d,1)+K*(zret(a:b,1) - H*xhat32(c:d,1));
          end
          Pk = Ppre;

elseif count == 1
    qsquared = 10000;
    K = [];
    I = eye(4);
    xhat32 = zeros(size(xhat));
    Pk = zeros(size(P));
    Mk = max(size(xhat));
    Mk = Mk/4;
    Qk = zeros(size(P));
    for k = 1:Mk
                    % INITIAL PREDICTION STEP ONLY!!:
        a = 2*k-1;
        b = 2*k;
        c = 4*k-3;
        d = 4*k;
        Qk(c:d,:) = qsquared*Q;
        Pk(c:d,:) = F*P(c:d,:)*F' + Qk(c:d,:);
        xhat32(c:d,1) = F*xhat(c:d,1);

                    %  UPDATE STEP:

        K = Pk(c:d,:)*H'*inv(H*Pk(c:d,:)*H' + R(a:b,:));  %  S1v = R
        Ppre(c:d,:) = (I-K*H)*Pk(c:d,:)*(I-K*H)'+K*R(a:b,:)*K';
        xhat33(c:d,1) = xhat32(c:d,1)+K*(zret(a:b,1) - H*xhat32(c:d,1));
     end
```

```
        Pk = Ppre;
     return
end
```

NAME	*constGKF*
INPUT	Q,P,R,xhat,zret,count
OUTPUT	xhat33,count
PURPOSE	Tracker for *fusim*, will perform the following functions: PREDICT, based on previous stuff (stored as globals) UPDATE, based on current measurement

```
%  CONSTANT GAIN KALMAN FILTER
function [xhat33,count] = constGKF(Q,P,R,xhat,zret,count)
global Kbar M F H;
if count > 1
        xhat32 = zeros(size(xhat));
        for k = 1:M
                        % PREDICTION STEP:
            a = 2*k-1;
            b = 2*k;
            c = 4*k-3;
            d = 4*k;
            xhat32(c:d,1) = F*xhat(c:d,1);

                        %  UPDATE STEP:
            xhat33(c:d,1) = xhat32(c:d,1)+Kbar(c:d,:)*( zret(a:b,1) - H*xhat32(c:d,1) );
        end
elseif count == 1
    r = zeros(2,2);
    Kbar = [];
    qsquared = 10000.0;
    Q = qsquared*Q;
    xhat32 = zeros(size(xhat));
    M = max(size(xhat));
    M = M/4;
    for k = 1:M
                        % PREDICTION STEP:
        a = 2*k-1;
        b = 2*k;
        c = 4*k-3;
        d = 4*k;
        r(1:2,1:2) = R(a:b,1:2);
        G = eye(4);
        [KBAR,Pbar] = dlqe(F,G,H,Q,r);
        Kbar = [Kbar;KBAR];
        xhat32(c:d,1) = F*xhat(c:d,1);

                        %  UPDATE STEP:
        xhat33(c:d,1) = xhat32(c:d,1)+KBAR*(zret(a:b,1) - H*xhat32(c:d,1) );
       end
end
```

NAME	*linkreport*
INPUT	xhat,simt
OUTPUT	linkdat,lstate
PURPOSE	This function creates a simulated data link report for the targets and allows for addition of attribute data, such as IFF data FORMAT: [x,y,velocity,heading,time,iff1,iff2,iff3,iff4,iffc,emitter1,emitter2,ID,reportingunit];

```
function [linkdat,lstate] = linkreport(xhat,simt)
global M;
linkdat = [];
lstate = [];
%tempxhat = [0;0];
for k = 1:M
   a = 4*k-3;  %1
   b = 4*k-2;  %2
```

```
    c = 4*k-1;  %3
    d = 4*k;    %4
 %   tempxhat(1) = xhat(a);
 %   tempxhat(2) = xhat(c);
    %  Add in noise
    x = xhat(a) + randn*50;
    y = xhat(c) + randn*50;
    vx = xhat(b) + randn*10;
    vy = xhat(d) + randn*10;
    %  Add in a set value of bias error, simulates DLRP/gridlock errors
    x = x + 100;  % 100 feet in bias error x-direction
    y = y + 100;  % 100 feet in bias error y-direction
    hdg = 0; %round(atan(vx,vy)*180/pi);
    v = sqrt(vx^2 + vy^2);
    linkvector = [x,vx,y,vy,v,hdg,simt,0,0,0,0,0,0];
    lstate = [lstate;linkvector];
    linkdat = [linkdat;x;y];
end
```

NAME	*simplot*
INPUT	mn,ns,xhat,Spst
OUTPUT	Plot with coordinates in feet
PURPOSE	This function is used to plot the main display (for each of the targets) Ownship is the first target.

```
function simplot(mn,ns,xhat,Spst)
%  mn -     Total number of targets
%  xhat -   Current tracking state.
global H;
xyplot = zeros(2,1);
figure(1)
axis([-250000 850000 -250000 1000000]);
xlabel('LONGITUDE - in feet for now'); ylabel('LATITUDE - in feet for now');
title(['Single Sensor Tracking Solutions']);
```

```
hold on;
pause(.01)
handle2 = [];
r=0; g=1; b=0;
handle1 = plot(Spst(1),Spst(3),'m*','erasemode','xor','markersize',6);
%set(handle1,'color',[r g b])

for k = 1:ns
   if k == 1
      r=0; g=0; b=1;
   elseif k == 2
      r=1; g=0; b=0;
   elseif k == 3
      r=.75; g=.5; b=0;
   elseif k == 4
      r=0; g=.5; b=.5;
   end

     for hh = 1:mn
        xyplot = H*xhat(4*hh-3:4*hh,:,k);
        handle2(hh) = plot(xyplot(1),xyplot(2),'+','erasemode','xor','markersize',6);
        set(handle2(hh),'color',[r g b])
        %plot(xyplot(1),xyplot(2),'*');
        %set(handle2,'xposition',xyplot(1),'yposition',xyplot(2));
     end

  end
```

NAME	*lsecalc*
INPUT	mn,state,posout,nsamples
OUTPUT	LSE
PURPOSE	This function calculates the Least Square Error comparing the state vector to the truth position.

```
function LSE = lsecalc(mn,state,posout,nsamples)
LSE = [];
for hh = 1:mn
  i1 = 4*hh-3;
  i2 = 4*hh-1;
  temp1 = [state(i1,:);
   state(i2,:)];

  tempposout = [posout(2*hh-1,2:(nsamples));posout(2*hh,2:(nsamples))];
  Kalmanerror = tempposout-temp1;
  temp2 = [];
  temp3 = 0.0;
  LSerror = [];
  for g = 1:nsamples-1,
   temp2 = [Kalmanerror(1,g);Kalmanerror(2,g)];
   temp3 = sqrt( temp2(1)^2 + temp2(2)^2 );
         LSerror = [LSerror,temp3];
  end
  LSE = [LSE;LSerror];
end
```

NAME	*errorplots*
INPUT	delta,merror,LSmean,trkr
OUTPUT	handl, error plots for each target and/or sensor
PURPOSE	This function plots the mean measurement errors and least squares tracking errors for each sensor and target

```
function handl = errorplots(delta,merror,LSmean,trkr)
%  merror is mean measurement error
%  statemean is the matrix of mean states
%  posout is the truth data
%  delta is the time step
%  nothing returned except the plot handle.
[a b c] = size(LSmean);
nsamples = max(size(merror));
time = [0:max(size(LSmean)-1)]*delta;
merror = merror(:,2:end,:);
   for hh = 1:a  %  Plot for each target:
      figure
      title(['Mean Error (-) vs. Time, 1 Sensor Type/4 Trackers/100 run(s)']);
      axis([0 900 -1 1]);
      xlabel('Time, (seconds)'); ylabel('Error, (feet)');
      hold on;
      %for k = 1:c
         plot(time,LSmean(hh,:,1),'b-');  %  PDA
         plot(time,LSmean(hh,:,2),'g-');  %  IMM
         plot(time,LSmean(hh,:,3),'r-');  %  KALMAN
         plot(time,LSmean(hh,:,4),'m-');  %  CGK
         plot(time,merror(hh,:,1),'c-');  %  Mean Error
         %end
   end


%for k = 1:c  %  Loop for each sensor
%    merror2 = merror(:,2:nsamples,k);
%    trkrID = trkr(k);
%    for hh = 1:a  %  Loop for each target
%       figure
%       plot(time,merror2(hh,:),'-',time,LSmean(hh,:,k),'-');
%       if trkrID == 1
%           title(['Mean Dist Error (-)/Mean PDAF Error (-) vs. Time (1 run(s))']);
%       elseif trkrID == 2
%           title(['Mean Dist Error (-)/Mean IMM Error (-) vs. Time (1 run(s))']);
%       elseif trkrID == 3
%           title(['Mean Dist Error (-)/Mean Kalman Error (-) vs. Time (1 run(s))']);
%       elseif trkrID == 4
%           title(['Mean Dist Error (-)/Mean Const Gain Kalman Error (-) vs. Time (1 run(s))']);
%       end
%    end
%end
handl = 0;
```

NAME	*trackplot*
INPUT	posout,zoutmean,statemean,sppos,spstatemean,linkmean
OUTPUT	hand, track plot
PURPOSE	This function plots the true position, the noisy measurements and the track states for each track and each sensor, and provides a single plot for comparison of all sensors and multiple plots of the primary tracker

```
function hand = trackplot(posout,zoutmean,statemean,sppos,spstatemean,linkmean)
 [mn lots ns] = size(statemean);
mn = mn/4;
%figure
%hold on
%po1 = zeros(2,max(size(posout)));  *******
%po2 = po1;  *********
%for i = 1:mn
%    po1(:,:) = posout(2*i-1:2*i,:);
%    plot(po1(1,:),po1(2,:),'-');
%end
%title(['True Target Motion/Mean Measured Position (1 run(s))']);
```

```
%for j = 1:ns
%    for i = 1:mn
%        po2(:,:) = zoutmean(2*i-1:2*i,:,j);
%        plot(po2(1,:),po2(2,:),'-');
%    end
%end

figure
hold on
Hline = [];
linecolor = [0 0 1];
for j = 1:ns  %  For each sensor:
   if j == 1
      r=0; g=0; b=1;
   elseif j == 2
      r=1; g=0; b=0;
   elseif j == 3
      r=.75; g=.5; b=0;
   elseif j == 4
      r=0; g=.5; b=.5;
   end
   for i = 1:mn %  Loop for each target:
      poo = statemean(4*i-3,:,j);  %  X values
      po1 = statemean(4*i-1,:,j);  %  Y values
      po2 = [poo;po1];
      Hline(i) = plot(po2(1,:),po2(2,:),'-');

   end
   set(Hline,'color',linecolor);
   linecolor = [r g b];
end
title(['Mean State (Tracking Solution) for All Sensors(100 runs)']);

%figure
axis([-250000 850000 -250000 1000000]);
```

```
xlabel('LONGITUDE (feet)'); ylabel('LATITUDE (feet)');
%hold on
plot(sppos(1,:),sppos(2,:),'-',spstatemean(1,:),spstatemean(3,:),'-' );
%poc = zeros(2,max(size(statemean)));
po1 = zeros(2,max(size(posout)-1));
for i = 1:mn  %  Plot actual for each target
   po1(:,:) = posout(2*i-1:2*i,2:end);
%    poc(:,:) = [statemean(4*i-3,:,1);statemean(4*i-1,:,1)];
   plot(po1(1,:),po1(2,:),'-');  %,poc(1,:),poc(2,:),'-' );
end
%  Plot the data link result:
for i = 1:mn
   link(:,:) = linkmean(2*i-1:2*i,1:end);
   plot(link(1,:),link(2,:),'m-');
end

%title(['Mean Tracker State/Link Report vs. True Position(1 run)']);
hand = 0;
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Chairman 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Captain Lee Lilly 1
   47123 Buse Rd Unit IPT
   Bldg 2272 Suite 455
   Patuxent River, MD 20670

5. Mr. Bill Hamel 1
   48142 Shaw Rd Unit 5
   Bldg 3197 Suite 1010
   Patuxent River, MD 20670

6. Dr. Rabinder Madan 1
   Office of Naval Research
   Balston Center Tower 1
   800 N. Quincy St
   Arlington, VA. 22217

7. Professor Russ Duren 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5107

8. Professor Gary Hutchins 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

9. Professor Bob Koyak 1
Department of Operations Research
Naval Postgraduate School
Monterey, CA 93943-5121

10. Mr. Mike Hinman 1
Air Force Research Laboratory (AFRL/IFEA)
32 Brooks Rd.
Rome, N.Y. 13441-4114

11. Mr. Oliver Drummond 1
10705 Cranks Road
Culver City, CA 90230

12. LCDR Dean Wilson... 1
44716 Emma Lane
Hollywood, MD 20636